
12.1. Presentación del capítulo

Este capítulo trata el proceso de la realización de caso de uso en el que modela interacciones entre objetos.

12.2. Actividad UP: Analizar un caso de uso

En capítulos anteriores ha visto cómo se produce el artefacto de clase de análisis de la actividad Analizar un caso de uso. El segundo artefacto que se produce por esta actividad es la realización de caso de uso, como se muestra en la figura 12.2. Tratamos las entradas a esta actividad en el capítulo 8.

Las clases de análisis modelan la estructura estática de un sistema y las realizaciones de caso de uso muestran cómo las instancias de las clases de análisis interactúan para realizar la funcionalidad del sistema. Esto es parte de la vista dinámica del sistema.

Sus objetivos para realización de caso de uso en análisis son los siguientes:

- Averiguar qué clases de análisis interactúan para realizar el comportamiento especificado por un caso de uso; puede descubrir nuevas clases de análisis a medida que realiza la realización de caso de uso.

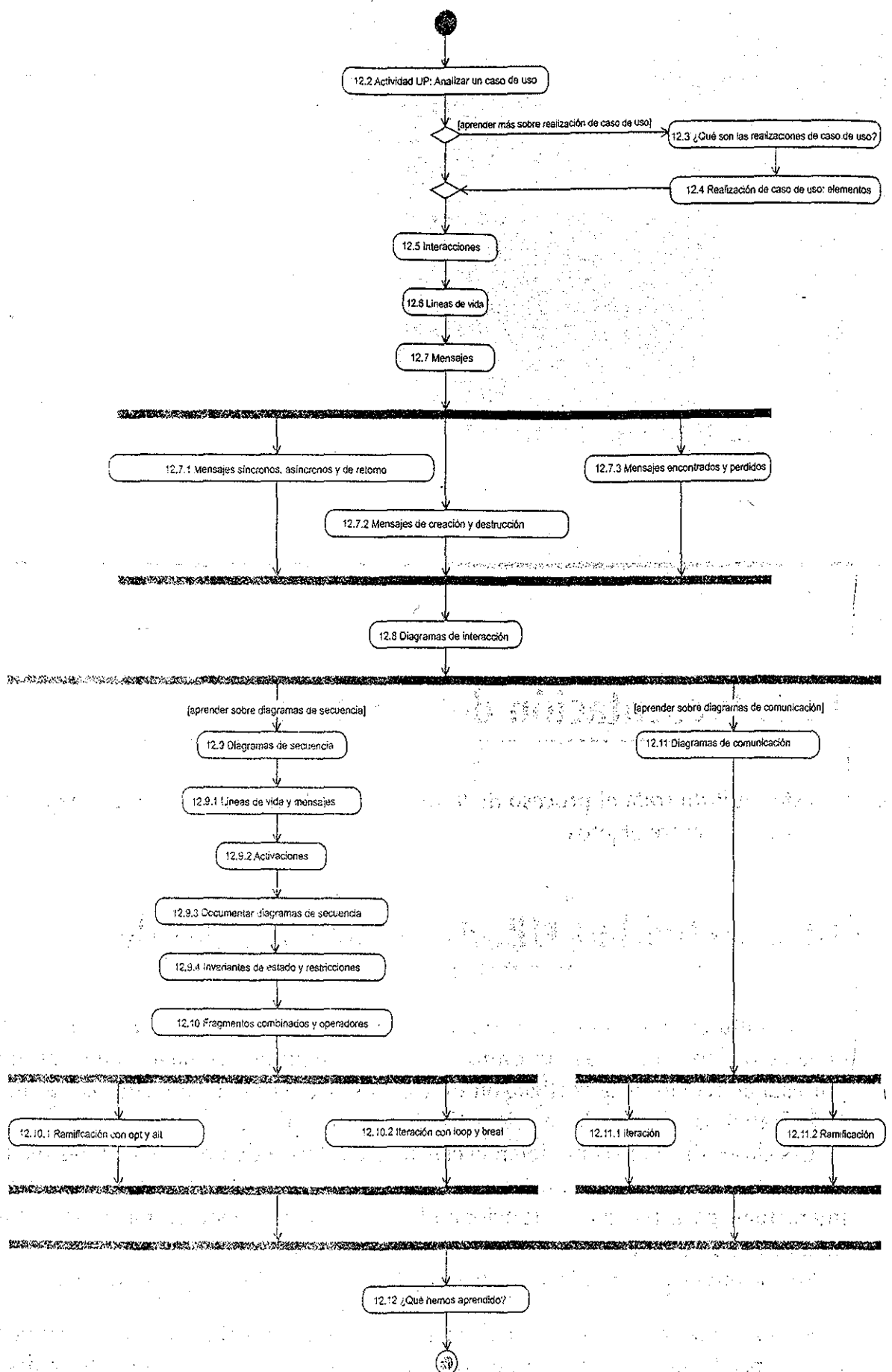


Figura 12.1.

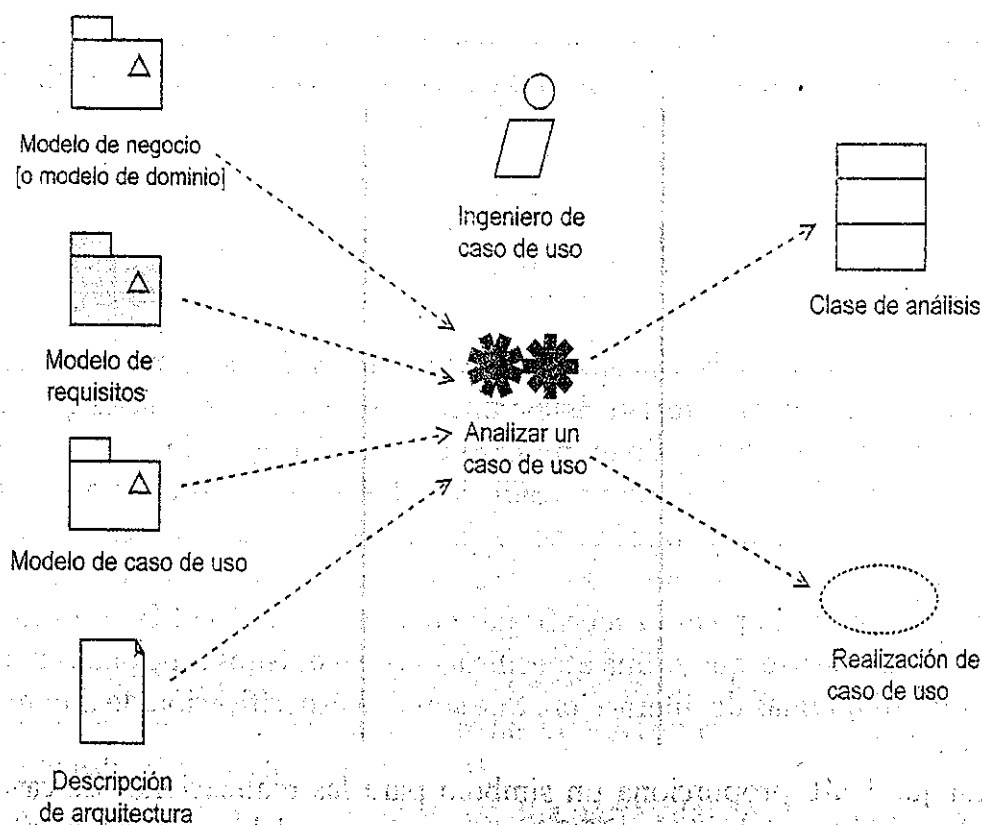


Figura 12.2. Adaptada de la figura 8.25 [Jacobson 1] con permiso de Addison-Wesley.

- Averiguar qué instancias de mensajes de esas clases se tienen que enviar entre sí para realizar el comportamiento especificado. Como verá en este capítulo, esto le dice:
 - Las operaciones clave que sus clases de análisis necesitan tener.
 - Los atributos clave de las clases de análisis.
 - Relaciones importantes entre clases de análisis:
- Actualizar su modelo de caso de uso, modelo de requisitos y clases de análisis con la información que obtiene de la realización de caso de uso. Mantenga los modelos coherentes entre sí.

En la realización de caso de uso en análisis, es esencial que se centre en capturar atributos, operaciones y relaciones clave entre clases de análisis. En este punto no está preocupado con detalles como parámetros de operación; descubrirá esta información en diseño.

Igualmente, no necesita crear una realización de caso de uso para todo caso de uso. Elija los casos de uso clave y trabaje en ellos. Mantenga la realización de casos de uso hasta que crea que tiene información suficiente para entender cómo funcionan conjuntamente las clases de análisis. Cuando tenga esta información, pare. UP es un proceso iterativo, por lo tanto, si decide que necesita realizar más trabajo sobre la realización de caso de uso más adelante, tendrá una oportunidad de hacerlo.

Al final de la realización de caso de uso en análisis, tendrá un modelo de análisis que proporciona una imagen de alto nivel del comportamiento dinámico del sistema.

12.3. ¿Qué son las realizaciones de caso de uso?

La clave para el análisis, después de encontrar las clases de análisis, es encontrar las realizaciones de caso de uso. Éstas constan de conjuntos de clases que realizan el comportamiento especificado en un caso de uso. Por ejemplo, si tiene un caso de uso PrestarLibro y ha identificado las clases de análisis Libro, Carnet y Prestatario y el actor Bibliotecario, necesita crear una realización de caso de uso que demuestre cómo estas clases y objetos de estas clases interactúan para realizar el comportamiento especificado en PrestarLibro. De esta forma, convierte un caso de uso, que es una especificación de requisitos funcional, en diagramas de clase y diagramas de interacción, que son una especificación de alto nivel de un sistema.

Aunque UML proporciona un símbolo para las realizaciones de caso de uso, como se muestra en la figura 12.3, raramente se modelan explícitamente. Esto es porque cada caso de uso tiene exactamente una realización de caso de uso, por lo tanto, no existe información adicional a capturar al crear un diagrama de realización de caso de uso. En su lugar, simplemente añade los elementos apropiados (véase tabla 12.1) a la herramienta de modelado y permite que las realizaciones de caso de uso sean una parte implícita del plano posterior del modelo.

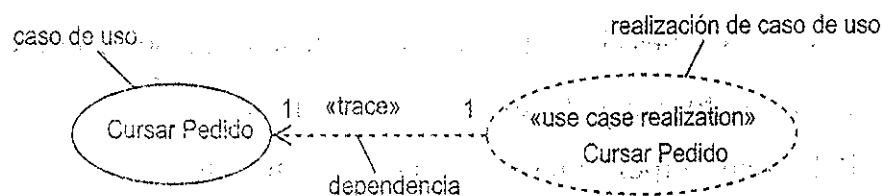


Figura 12.3.

Tabla 12.1.

Elemento	Finalidad
Diagramas de clase de análisis.	Muestran las clases de análisis que interactúan para realizar el caso de uso.
Diagramas de interacción.	Muestra interacciones entre instancias específicas que realizan el caso de uso; son "instantáneas" del sistema que se ejecuta.
Requisitos especiales.	El proceso de realización de caso de uso puede descubrir nuevos requisitos específicos al caso de uso; estos se deben capturar.

Elemento	Finalidad
Mejora del caso de uso.	Se puede descubrir nueva información durante la realización, esto significa que el caso de uso original se tiene que actualizar.

12.4. Realización de caso de uso: elementos

Las realizaciones de caso de uso constan de los elementos que se muestran en la tabla 12.1. La realización de caso de uso es fundamentalmente un proceso de mejora. Toma una especificación de un aspecto del comportamiento del sistema según se captura en un caso de uso y cualquier requisito asociado y modela cómo se puede realizar esto por interacciones entre instancias de las clases de análisis que ha identificado. Va de una especificación general de un comportamiento requerido a una descripción bastante detallada de las interacciones entre clases y objetos que harán que este comportamiento sea real.

Los diagramas de clase de análisis son una parte vital de una realización de caso de uso. Debería "contar una historia" sobre el sistema, sobre cómo un conjunto de clases están relacionadas de modo que esas instancias de esas clases pueden colaborar para realizar el comportamiento especificado en uno (o más) casos de uso.

Al igual que diagramas de clase de análisis, puede crear diagramas que demuestran explícitamente cómo las instancias de esas clases de análisis colaboran e interactúan para realizar algunos o todos los comportamientos de caso de uso. Estos diagramas se conocen como diagramas de interacción y existen cuatro tipos: diagramas de secuencia, diagramas de comunicación, diagramas de visión de interacción y diagramas de tiempo. En este capítulo vemos los diagramas de secuencia y comunicación, los diagramas de visión de interacción se ven en el capítulo 15 y los diagramas de tiempo en el capítulo 20.

El modelado orientado a objetos es un proceso iterativo por lo que no debería estar demasiado sorprendido si descubre nuevos requisitos o si necesita modificar casos de usos existentes una vez que empieza a modelar en más profundidad. Todo esto es parte de la realización de caso de uso; debe mantener los documentos existentes actualizados a medida que descubre más información sobre el sistema. Como tal, debe actualizar el modelo de caso de uso, el modelo de requisitos y las clases de análisis para hacer que todas sean coherentes.

12.5. Interacciones

Las interacciones son sencillas unidades de comportamiento de un clasificador. Este clasificador, conocido como el clasificador de contexto, proporciona el contexto para la interacción.

Una interacción puede utilizar cualquiera de las características de su clasificador de contexto o cualquier característica a la que tenga acceso el clasificador de contexto (por ejemplo, variables temporales o globales).

En la realización de caso de uso, el clasificador de contexto es un caso de uso y crea una o más interacciones para demostrar cómo el comportamiento especificado por el caso de uso se puede realizar por instancias de clasificadores (en este caso, clase de análisis) pasando mensajes de un lado a otro.

A medida que trabaja en los diagramas de interacción, empieza a descubrir cada vez más operaciones y atributos de las clases de análisis. Los diagramas de clases de análisis se deberían actualizar con esta información como parte del proceso de realización del caso de uso. Los elementos clave en diagramas de interacción son líneas de vida y mensajes. Veremos esto en detalle en los siguientes apartados.

12.6. Líneas de vida

Una línea de vida representa un solo participante en una interacción, es decir, representa cómo una instancia de un clasificador específico participa en la interacción. Las sintaxis de la línea de vida se resumen en la figura 12.4.

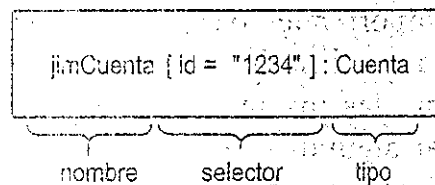


Figura 12.4.

Toda línea de vida tiene un nombre opcional, un tipo y un selector opcional.

- **Nombre:** Utilizado para hacer referencia a la línea de vida dentro de la interacción.
- **Tipo:** El nombre del clasificador del que la línea de vida representa una instancia.
- **Selector:** Una condición booleana que se puede utilizar para seleccionar una sola instancia que satisface la condición. Si no existe ningún selector, una línea de vida hace referencia a una instancia arbitraria del clasificador. Los selectores son solamente válidos si el tipo tiene una multiplicidad mayor que uno de modo que existen muchas instancias entre las que elegir. En la figura 12.4 el selector selecciona una instancia de Cuenta que tiene un id de "1234".

Las líneas de vida se dibujan con el mismo icono que su tipo y tienen una línea vertical discontinua cuando se utiliza en diagramas de secuencia. Algunos ejemplos de líneas de vida se muestran en la figura 12.5.

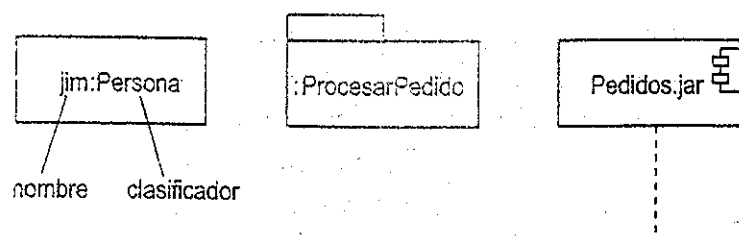


Figura 12.5.

Puede pensar en una línea de vida que representa cómo una instancia del clasificador puede participar en la interacción. Sin embargo, no representa ninguna instancia particular del clasificador. Esta es una distinción sutil pero importante. La interacción describe cómo las instancias del clasificador interactúan de una forma general, en lugar de especificar solamente una interacción particular entre un conjunto de instancias particulares. Puede pensar en una línea de vida como que representa un rol que una instancia del clasificador puede desempeñar en la interacción.

Puede mostrar instancias verdaderas directamente en un diagrama de interacción si lo desea. Simplemente utilice la notación normal de instancia; es decir, el símbolo del clasificador con el nombre de la instancia, selector (si alguno), dos puntos y el nombre del clasificador, todo subrayado.

Esta distinción entre líneas de vida e instancias, da paso a dos formas diferentes de diagramas de interacción. Un diagrama de interacción de forma genérica muestra la interacción entre líneas de vida que representa instancias arbitrarias. Un diagrama de interacción de forma de instancia muestra la interacción entre instancias determinadas. Los diagramas de forma genérica tienden a ser los más comunes y de más utilidad.

Para completar la interacción, necesita especificar mensajes que se envían entre las líneas de vida. Examinamos los mensajes en el siguiente apartado.

12.7. Mensajes

Un mensaje representa un tipo específico de comunicación entre dos líneas de vida en una interacción. Esta comunicación puede implicar:

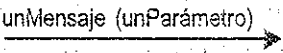
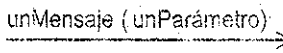

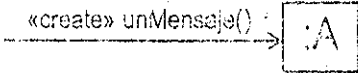
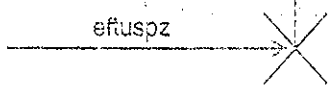

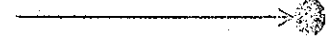
- Invocar una operación; un mensaje de llamada.
- Crear o destruir una instancia; un mensaje de creación o destrucción.
- Enviar una señal.

Cuando una línea de vida recibe un mensaje de llamada, éste es una petición para la invocación de una operación que tiene la misma firma que el mensaje. Por lo tanto, para cada mensaje de llamada recibido por una línea de vida, debe hacer una operación correspondiente en el clasificador de esa línea de vida. UML permite que los mensajes en diagramas de interacción no vayan al paso de las operaciones para que pueda trabajar en el modelo dinámica y flexiblemente. Sin embargo, en un análisis posterior, tienen que ir al mismo ritmo.

Cuando una línea de vida ejecuta un mensaje, tiene foco de control o activación. A medida que progresa la interacción con el tiempo, la activación se mueve entre las líneas de vida; este movimiento se denomina el flujo de control.

Los mensajes se dibujan como flechas entre líneas de vida. Si la línea de vida tiene una línea vertical discontinua (como en los diagramas de secuencia), los mensajes generalmente se dibujan entre esas líneas. De lo contrario los mensajes se dibujan entre los cuadros de las líneas de vida; verá muchos ejemplos de esto en breve. Existen siete tipos de mensajes, según se ilustra en la tabla 12.2.

Tabla 12.2.

Símbolos	Nombre	Significado
	Mensaje síncrono	El emisor espera hasta que el receptor regresa de ejecutar el mensaje.
	Mensaje asíncrono	El emisor envía el mensaje y continúa ejecutando; no espera una respuesta del receptor.
	Retorno de mensaje	El receptor de un mensaje anterior devuelve el foco del control al emisor de ese mensaje.
	Creación de objeto	El emisor crea una instancia del clasificador especificado por el receptor.
	Destrucción de objeto	El emisor destruye el receptor. Si la línea de vida tiene una línea vertical discontinua, se termina con un X.
	Mensaje encontrado	El emisor del mensaje está fuera del ámbito de la interacción. Utilice esto cuando quiera mostrar un recibo de mensaje, pero no quiera mostrar de dónde procede.
	Mensaje perdido	El mensaje nunca llega a su destino. Se puede utilizar para indicar condiciones de error en los que los mensajes se pierden.

12.7.1. Mensajes síncronos, asíncronos y de retorno

En una llamada de mensaje síncrono, el emisor espera a que el receptor acabe de ejecutar la operación solicitada. En una llamada de mensaje asíncrono, el emisor no

espera sino que continúa al siguiente paso. En los modelos de análisis, la distinción entre mensajes síncronos y asíncronos normalmente implica un alto nivel de detalle. En análisis, no está preocupado por la semántica detallada del envío de mensajes, sino solamente con el hecho de que el mensaje se envía. Como tal, puede mostrar todos los mensajes como síncronos o asíncronos; en realidad no importa. Nuestra preferencia es mostrar todos los mensajes como síncronos porque éste es el caso más restringido. Los mensajes síncronos indican una secuenciación estricta de llamadas de operación, mientras que los mensajes asíncronos indican la posibilidad de concurrencia.

En diseño, puede ser importante distinguir entre esos mensajes que son síncronos y los que son asíncronos para que pueda diseñar flujos concurrentes de control.

Puede mostrar o no, según elija, mensajes de retorno en realizaciones de caso de uso a nivel de análisis. Por lo general no son tan importantes. Tendemos a mostrarlos si no saturan el diagrama.

12.7.2. Mensajes de creación y destrucción

En el análisis orientado a objetos, generalmente no necesita preocuparse por la semántica exacta de la creación o destrucción de objetos, pero es importante que entienda lo que sucede, por lo que tratamos este tema aquí.

El mensaje de creación de objeto siempre se dibuja como una línea continua con una punta de flecha abierta. Puede mostrar creación de objetos con sólo enviar un mensaje estereotipado `<<create>>`, o bien puede enviar un mensaje de creación de objeto que también puede estereotipar `<<create>>`. En C++, C# o Java, las operaciones de creación de objeto son operaciones especiales conocidas como constructores; éstas tienen el mismo nombre que la clase del objeto, no valor de retorno y cero o más parámetros. Por lo tanto, por ejemplo, si quisiera crear un nuevo objeto Cuenta, podría enviar un mensaje denominado Cuenta() para enviar un objeto Cuenta e inicializar su atributo númeroCuenta en algún valor. Sin embargo, no todos los lenguajes orientados a objetos tienen constructores; en Smalltalk por ejemplo, probablemente enviaría el mensaje `<<create>> init:numeroCuenta`.

La destrucción de objeto se muestra como una línea continua con una punta de flecha abierta estereotipada `<<destroy>>`. Destrucción significa que la instancia de clasificador a la que hace referencia la línea de vida destino ya no se encuentra disponible para su uso. Si la línea de vida tiene una línea vertical discontinua, debe terminarla con una gran cruz en el punto de destrucción. No existen valores de retorno de la destrucción de objeto.

Lenguajes diferentes tienen diferente semántica de destrucción. Por ejemplo, en C++, la destrucción se gestiona explícitamente por el programador y cuando se destruye un objeto, se invoca (si existe) un método especial denominado destructor. Este método se utiliza a menudo para realizar actividades de limpieza. Después de invocar al destructor, se libera la asignación de memoria del objeto.

En lenguajes como Java y C#, la destrucción de objeto se gestiona por la máquina virtual por medio de una estrategia denominada recolección de basura. Por ejem-

plo, cuando un objeto en un programa Java ya no está referenciado por ningún otro objeto, se marca como listo para destrucción. La destrucción ocurrirá en algún momento futuro de acuerdo al algoritmo de la recolección de basura, pero no sabe cuándo sucederá. Los objetos Java y C# pueden tener un método "finalizar" que se ejecutará en el momento de la destrucción por el recolector de basura. Sin embargo, este método es peligroso de utilizar ya que no sabe cuándo lo invocará el recolector de basura.

12.7.3. Mensajes encontrados y perdidos

Por lo general puede ignorar los mensajes encontrados y perdidos en análisis. Los incluimos aquí principalmente para completar la información.

Los mensajes encontrados pueden ser de utilidad si necesita mostrar un recibo de mensaje por una clase, pero no sabe (en ese momento en el tiempo) dónde se originó ese mensaje. Encontramos que esto no sucede demasiado a menudo en la práctica.

Los mensajes perdidos le permiten mostrar que un mensaje se ha perdido; nunca llega a su destino. Esto podría ser de utilidad en diseño para mostrar cómo los mensajes se podrían perder durante una condición de error. Sin embargo, nunca hemos sentido la necesidad de utilizarlo.

12.8. Diagramas de interacción

Los diagramas de interacción de UML se pueden utilizar para modelar cualquier tipo de interacción entre instancias de clasificador. En la realización de caso de uso, se utilizan específicamente para modelar interacciones entre objetos que realizan un caso de uso o parte de un caso de uso. Existen cuatro tipos de diagramas de interacción, cada uno de los cuales enfatiza un aspecto diferente de la interacción.

- **Diagramas de secuencia:** Éstos enfatizan la secuencia de envíos de mensajes ordenada en el tiempo entre líneas de vida. Los usuarios tienden a entender los diagramas de secuencia mejor que los diagramas de comunicación ya que son más sencillos de leer. Los diagramas de comunicación tienen una tendencia a saturarse muy rápidamente. Tratamos los diagramas de secuencia en el siguiente apartado.
- **Diagramas de comunicación:** Enfatizan las relaciones estructurales entre objetos y son muy útiles en análisis, especialmente para crear un rápido bosquejo de una colaboración de objeto. En UML 2, estos diagramas ofrecen solamente un subconjunto de la funcionalidad de los diagramas de secuencia. Tratamos los diagramas de comunicación más adelante.
- **Diagramas de visión de interacción:** Muestran lo complejo que se realiza el comportamiento por un conjunto de interacciones sencillas. Éstos son un

caso especial de diagramas de actividad en el que los nodos hacen referencia a otras interacciones. Son de utilidad para modelar el flujo de control dentro de un sistema. Tratamos los diagramas de visión de interacción en el capítulo 15.

- **Diagramas de tiempo:** Enfatizan los aspectos en tiempo real de una interacción. Su finalidad principal es ayudarle a razonar sobre el tiempo. Tratamos los diagramas de tiempo en el capítulo 20.

Los diagramas de secuencia y comunicación son los diagramas más importantes desde la perspectiva de la realización de caso de uso y los examinamos en detalle en el resto de este capítulo.

12.9. Diagramas de secuencia

Los diagramas de secuencia muestran interacciones entre líneas de vida como una secuencia ordenada en el tiempo de eventos. Son la forma de diagrama de interacción más flexible.

Cuando modela, normalmente empieza esbozando una realización de caso de uso, utilizando un diagrama de comunicación porque es más sencillo situar líneas de vida en el diagrama y conectarlas. Sin embargo, cuando necesita centrarse en la secuenciación real de los eventos, siempre es mucho más sencillo utilizar un diagrama de secuencia.

12.9.1. Líneas de vida y mensajes

Para investigar las líneas de vida y mensajes, tomamos un ejemplo de un sencillo sistema de registro de cursos. Considere realizar el caso de uso `AñadirCurso` mostrado en la figura 12.6. Hemos mantenido este caso de uso en un nivel muy alto para proporcionar un ejemplo sencillo.

El análisis inicial del caso de uso ha creado el diagrama de clase de análisis de alto nivel en la figura 12.7. Dada la especificación de caso de uso y el diagrama de clase, dispone de suficiente información para crear un diagrama de secuencia.

La figura 12.8 muestra un diagrama de secuencia que realiza el comportamiento especificado por el caso de uso `AñadirCurso`. Según la especificación UML 2, los nombres del diagrama de interacción pueden estar prefijados por `sd` para indicar que el diagrama es un diagrama de interacción. Aunque parezca extraño, `sd` se utiliza como un prefijo para todos los tipos de diagramas de interacción, no solamente los diagramas de secuencia.

En este punto, merece la pena recordarle que cuando empieza crear diagramas de secuencia como parte de la realización del caso de uso, es posible que encuentre que necesita modificar el diagrama de clase de análisis o incluso el caso de uso. No hay problema, todo ello es parte del proceso de análisis. El caso de uso, el diagrama de clase de análisis y el diagrama de secuencia, todos evolucionan juntos con el tiempo.

Caso de uso: AñadirCurso	
ID:	8
Breve descripción:	Añadir detalles de un nuevo curso al sistema.
Actores principales:	Secretario
Actores secundarios:	Ninguno
Precondiciones:	1. El Secretario se ha conectado al sistema.
Flujo principal:	1. El Secretario selecciona "añadir curso". 2. El Secretario escribe el nombre del nuevo curso. 3. El sistema crea el nuevo curso.
Postcondiciones:	1. Un nuevo curso se ha añadido al sistema.
Flujos alternativos:	CursoYaExiste

Figura 12.6.

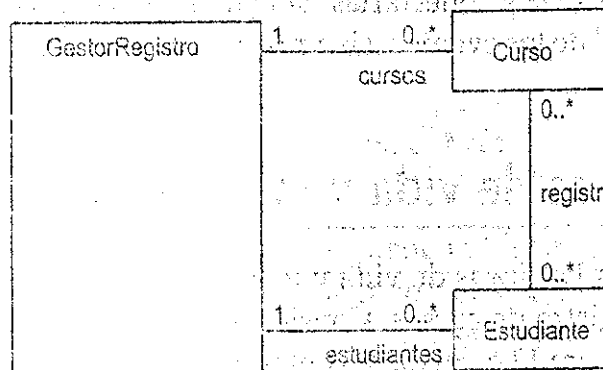


Figura 12.7.

Considere el diagrama de secuencia en la figura 12.8. Los diagramas de secuencia se ejecutan de arriba a abajo y las líneas de vida de izquierda a derecha. Las líneas de vida se sitúan horizontalmente para minimizar el número de líneas que se cruzan en el diagrama y se sitúan verticalmente de acuerdo a cuándo se crean. Por debajo de cada línea de vida se encuentra una línea continua que indica la duración de la línea de vida con el tiempo.

Observe que la figura 12.8 muestra cómo se realiza el comportamiento del caso de uso, pero no es una representación exacta de cada paso en el caso de uso. Éste es un punto importante. Los pasos 1 y 2 del caso de uso implican cierto tipo de interfaz de usuario que no veremos hasta el diseño, por lo que lo hemos omitido del diagrama de secuencia. En diseño, podemos añadir una capa de interfaz de usuario al diagrama de secuencia que ayude a aclarar las cosas. En análisis, solamente estamos preocupados por capturar el comportamiento esencial de las clases de análisis.

importante

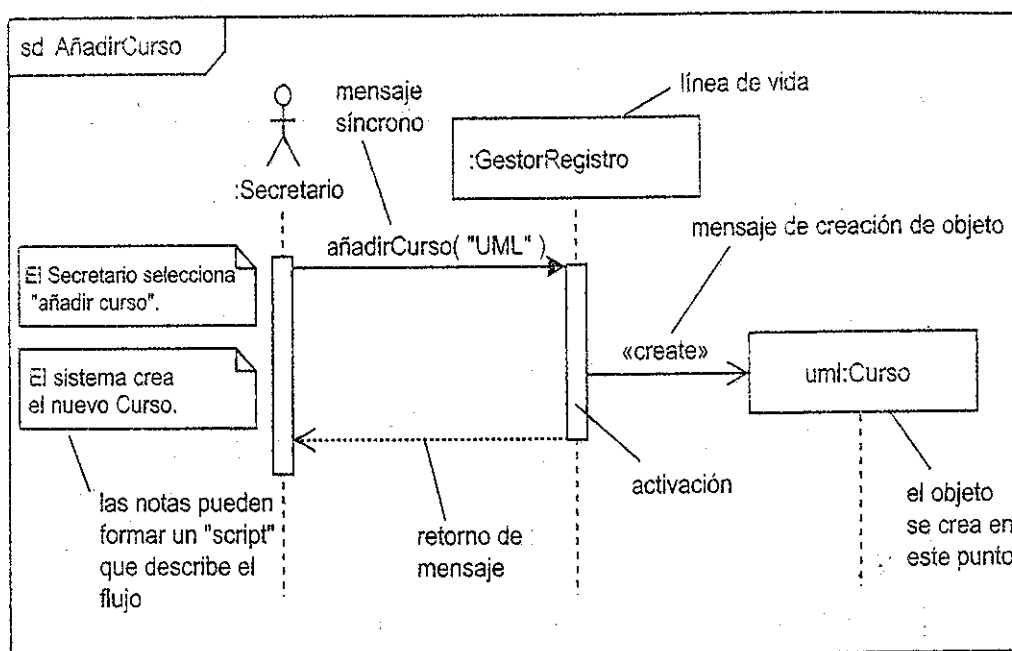


Figura 12.8.

Examinemos otro ejemplo de caso de uso del sistema de registro de cursos, **EliminarCurso** (véase la figura 12.9).

Caso de uso: EliminarCurso
ID: 8
Breve descripción: Eliminar un curso del sistema.
Actores principales: Secretario
Actores secundarios: Ninguno.
Precondiciones: 1. El Secretario se ha conectado al sistema.
Flujo principal: 1. El Secretario selecciona "eliminar curso". 2. El Secretario escribe el nombre del curso. 3. El sistema elimina el curso.
Postcondiciones: 1. Un curso se ha eliminado del sistema.
Flujos alternativos: CursoNoExiste

Figura 12.9.

En este caso de uso estamos destruyendo un objeto. Para indicar la destrucción de objeto, termine la línea de vida con una cruz grande como se muestra en la figura 12.10. Si no sabe cuándo se destruye el objeto o no le importa, simplemente termine la línea de vida de forma normal.

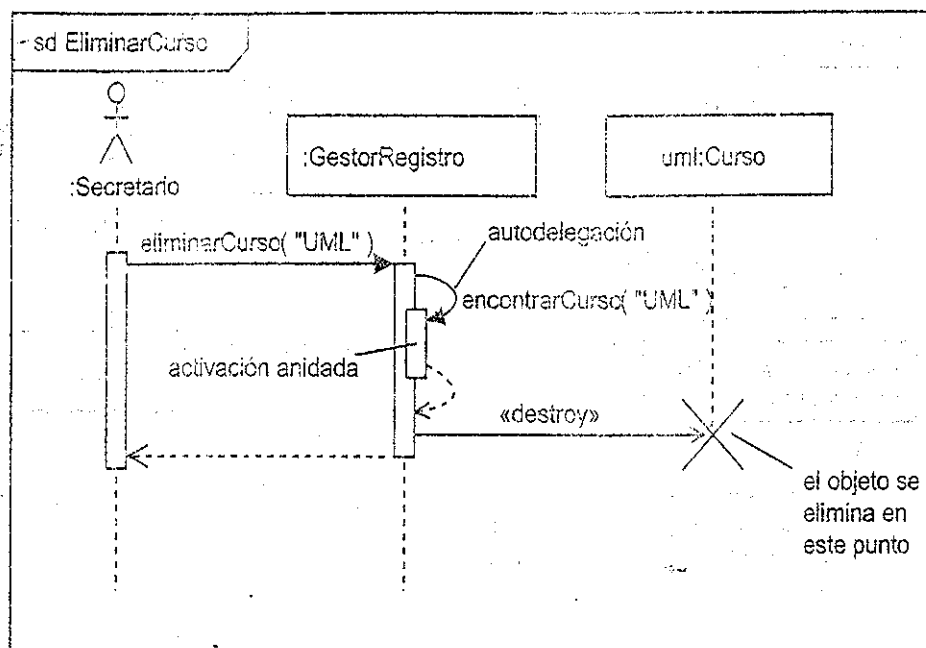


Figura 12.10.

La figura 12.10 también muestra la autodelegación, es decir, una línea de vida se envía un mensaje a sí misma. Esto crea una activación anidada (véase el siguiente apartado). La autodelegación es común en sistema orientado a objetos. Los objetos ofrecen un conjunto de servicios públicos (las operaciones públicas) que se pueden invocar por objetos cliente; pero, por lo general, también tienen un conjunto de operaciones privadas de "ayuda" que están específicamente diseñadas para ser invocadas por el propio objeto. En este ejemplo, la línea de vida :gestorRegistro se envía a sí misma el mensaje encontrarCurso ("UML") para encontrar un objeto curso UML si existe alguno. Las operaciones privadas de un objeto solamente se pueden invocar por ese propio objeto por medio de una autodelegación.

12.9.2. Activaciones

Sitúe rectángulos largos y estrechos en la línea discontinua debajo de la línea de vida para indicar cuándo una determinada línea de vida tiene el foco de control. Estos rectángulos se denominan activaciones o foco de control.

Hemos observado que en *The Unified Modeling Language Reference Manual, Second Edition* [Rumbaugh 1] hace referencia a la activación como "un término UML 1 reemplazado por especificación de la ejecución". Sin embargo, en el momento de ir a imprenta, este término no aparece en *Unified Modeling Language: Superstructure, version 2.0* [UML2S], mientras que sí lo hacen "activación" y "foco de control". Podemos concluir que "activación" y "foco de control" son los términos estándar. Lo mencionamos aquí porque puede encontrarse el término "especificación de ejecución" como un sinónimo de activación debido a su inclusión en [Rumbaugh 1].

En la figura 12.8, el actor :Secretario empieza con el foco de control. Envía el mensaje añadirCurso("UML") a :GestorRegistro que invoca su operación añadirCurso(...) con el parámetro "UML". Durante la ejecución de esta opera-

ción, :GestorRegistro tienen el foco de control. Sin embargo, observe que este foco de control está anidado dentro del foco de control del actor :Secretario. Esto es bastante normal, un objeto empieza con el foco de control e invoca una operación sobre otro objeto anidando el foco de control. Este objeto puede entonces invocar una operación sobre otro objeto anidando aún más el foco de control.

Dentro de la ejecución de la operación `añadirCurso(...)`, :GestorRegistro crea un nuevo objeto, el objeto `uml:Curso`.

Las activaciones parecen no utilizarse en los últimos años y encontrará que los modeladores no se preocupan de mostrarlas. Esto es en parte porque algunas herramientas UML no soportan activaciones muy bien y, en parte, porque no son tan importantes, especialmente en modelos de análisis. En un sistema orientado a objetos que se ejecuta, las activaciones se ocupan de sí mismas por medio de la semántica normal de invocación de operación. De hecho, diagramas complejos de secuencia pueden ser más sencillos de leer sin las activaciones. Nuestro estilo es utilizar activaciones a menos que hagan que el diagrama sea difícil de leer.

12.9.3. Documentar diagramas de secuencia

Una característica bastante atractiva de los diagramas de secuencia es que puede añadir un "script" a un diagrama al situar notas debajo de la parte izquierda (véase la figura 12.8). Esto hace que el diagrama sea mucho más accesible a los grupos de decisión no técnicos, como usuarios y expertos del negocio. El script puede constar de los pasos reales de un caso de uso o simplemente un resumen textual de lo que sucede en el diagrama. En cualquier caso, un script puede ser una incorporación de utilidad a un diagrama de secuencia, especialmente si el diagrama es complejo.

12.9.4. Invariantes de estado y restricciones

Cuando una instancia recibe un mensaje, puede hacer que cambie de estado.

Un estado se define como "una condición o situación durante la vida de un objeto durante la que satisface alguna condición, realiza alguna actividad o espera algún evento" [Rumbaugh 1]. Todo clasificador puede tener una máquina de estado que describe el ciclo de vida de sus instancias en términos de los estados en los que pueden estar y los eventos que hacen que pasen entre esos estados.

No todos los mensajes causan un cambio de estado. Por ejemplo, un mensaje que simplemente devuelve el valor de algún atributo y que no tiene efectos secundarios, nunca genera un cambio de estado. Puede mostrar el estado de las instancias en las líneas de vida al utilizar invariantes de estado. Añadir invariantes de estado a un diagrama de secuencia puede ser una técnica de análisis de mucha utilidad porque le permite capturar los estados clave en el ciclo de vida de una línea de vida. Estos estados indican estados importantes del sistema y pueden ser la base de las máquinas de estado que tratamos en el capítulo 21.

Examinemos un ejemplo específico: la figura 12.11 muestra un caso de uso de un sistema de procesamiento de pedidos que está sujeto a las siguientes restricciones:

- El pedido se debe pagar en su totalidad con un solo pago.
- Los artículos especificados en el pedido solamente se pueden entregar después del pago.
- Los artículos se entregan al cliente a los 28 días del pago.

Caso de uso: ProcesarUnPedido	
ID:	5
Breve descripción:	El Cliente realiza un pedido que luego se paga y se entrega.
Actores principales:	Cliente
Actores secundarios:	Ninguno.
Precondiciones:	Ninguna.
Flujo principal:	<ol style="list-style-type: none"> 1. El caso de uso empieza cuando el actor Cliente crea un nuevo pedido. 2. El Cliente paga el pedido en su totalidad. 3. Los artículos se entregan al Cliente a los 28 días de la fecha del pago final.
Postcondiciones:	<ol style="list-style-type: none"> 1. El pedido se ha pagado. 2. Los artículos se han entregado a los 28 días del último pago.
Flujos alternativos:	PagoSuperior PedidoCancelado ArtículosNoEntregados ArtículosEntregadosTarde PagoParcial

Figura 12.11.

El procesamiento de pedido del mundo real es normalmente más complejo y flexible que esto. No obstante, el ejemplo sirve para mostrar los invariantes de estado. La figura 12.12 muestra un diagrama de secuencia para `ProcesarUnPedido`.

Puede ver que cuando se crea una instancia `Pedido`, inmediatamente pasa al estado impagado (dibujado como un rectángulo redondeado). Esto le dice que todos los `Pedidos` están creados en el estado impagado. Al recibir el mensaje `aceptarPago()` que debe ser para un pago completo, la instancia `Pedido` pasa al estado pagado. En algún momento más adelante, a la instancia `GestorEntrega` se le envía el mensaje `entregar()`. Reenvía este mensaje a la instancia `Pedido` haciendo que pase al estado entregado.

Si la clase `Pedido` también tiene una máquina de estado, los estados en esa máquina deben corresponder a cualquier invariante de estado que pueda tener en sus diagramas de secuencia.

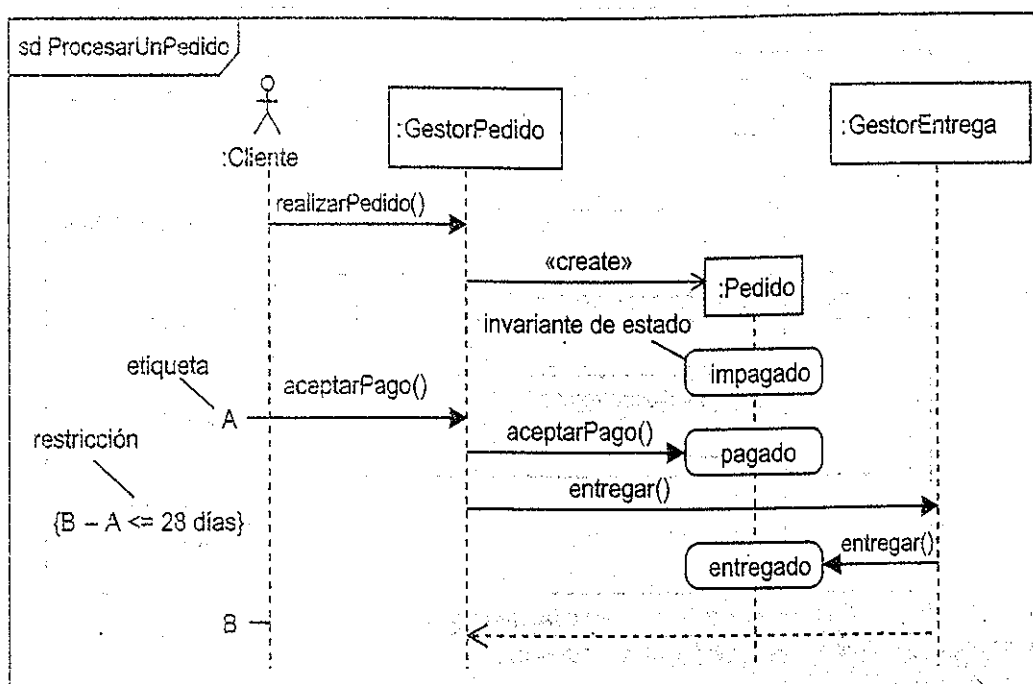


Figura 12.12.

La figura 12.12 ilustra el uso de restricciones que se escriben entre llaves y se sitúan en (o cerca de) las líneas de vida. Una restricción situada en una línea de vida indica algo que debe ser verdadero sobre instancias desde ese punto en adelante.

Las restricciones a menudo se indican en lenguaje informal, aunque UML tiene un lenguaje formal de restricción (OCL), que tratamos en el capítulo 25.

La figura muestra una restricción de duración. La línea de vida `:Cliente` tiene dos etiquetas, A y B, y una restricción $\{B - A \leq 28 \text{ días}\}$. Esto dice que la distancia en tiempo entre los puntos A y B debe ser menor que o igual a 28 días. El punto A marca el punto en el que se debe realizar el pago y el punto B marca el punto en el que los productos se envían al cliente. Es decir, la restricción significa que "el pedido se debería entregar a no más de 28 días después de recibir el pago".

Cualquier tipo de restricción se puede situar en una línea de vida. Las restricciones que restringen valores de atributo de instancias son bastante comunes.

Por último, observe que en la figura 12.12 no hemos utilizado ni activaciones ni retornos de mensaje. Esto es porque el énfasis en este diagrama está en la duración y en los invariantes de estado y estas características no añadirían nada.

12.10. Fragmentos combinados y operadores

Los diagramas de secuencia se pueden dividir en áreas denominadas fragmentos combinados. La figura 12.13 ilustra la sintaxis de fragmento combinado, que es bastante completa. Todo fragmento combinado tiene un operador, uno o más operandos y cero o más condiciones de protección. El operador determina cómo se ejecutan sus operandos.

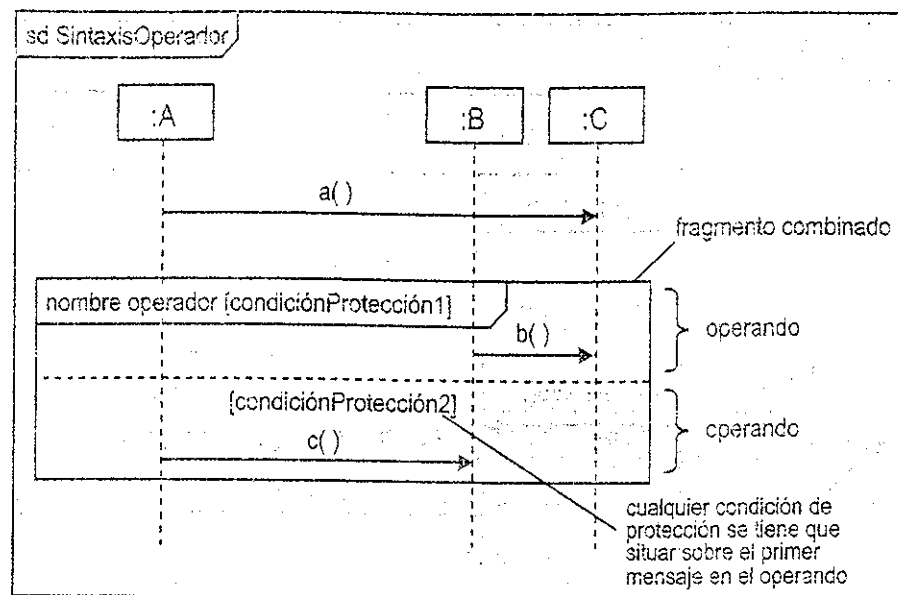


Figura 12.13.

Las condiciones de protección determinan si sus operandos se ejecutan. Las condiciones de protección son expresiones booleanas, y el operando se ejecuta si y sólo si la expresión evalúa en verdadero. Una sola condición de protección se puede aplicar a todos los operandos, o cada operando puede tener su propia condición de protección única. La lista completa de operadores se proporciona en la tabla 12.3.

Tabla 12.3.

Operador	Nombre	Semántica
opt	opción	Existe un solo operando que se ejecuta si la condición es verdadera (como if... then).
alt	alternativas	Se ejecuta el operando cuya condición es verdadera. La palabra clave <i>else</i> se puede utilizar en lugar de una expresión booleana (como select... case).
loop	bucle	Esto tiene una sintaxis especial: loop min, max [condición].
break	saltar	Si la condición de protección es verdadera, el operando se ejecuta, no el resto de la interacción que engloba.
ref	referencia	El fragmento combinado hace referencia a otra interacción.
par	paralelo	Todos los operandos se ejecutan en paralelo.
critical	crítico	El operando se ejecuta sin interrupción.
seq	secuenciación débil	Todos los operandos se ejecutan en paralelo sujetos a la siguiente restricción: los eventos que

Operador	Nombre	Semántica
		lleguen en la misma línea de vida de diferentes operandos, ocurren en la misma secuencia que los operandos. Esto da lugar a una forma de secuenciación débil, de ahí el nombre.
strict	secuenciación estricta	Los operandos se ejecutan en secuencia estricta.
neg	negativa	El operando muestra interacciones inválidas. Utilice esto cuando quiera mostrar interacciones que deben no suceder.
ignore	ignorar	Lista mensajes que se omiten intencionalmente de la interacción; los nombres de los mensajes ignorados están situados entre llaves en una lista delimitada por comas después del nombre del operador, como {m1, m2, m3}. Por ejemplo, una interacción podría representar un caso de prueba en el que elige ignorar alguno de los mensajes.
consider	considerar	Lista mensajes que se han incluido intencionalmente en la interacción. Los nombres de los mensajes están situados entre llaves, en una lista separados por comas después del nombre del operador. Por ejemplo, una interacción podría representar un caso de prueba en el que elija incluir un subconjunto del conjunto de mensajes posibles.
assert	aserción	El operando es el único comportamiento válido en ese punto en la interacción y cualquier otro comportamiento tendría un error. Utilice esto como una forma de indicar que cierto comportamiento debe ocurrir en cierto momento en la interacción.

Los operadores más importantes son opt, alt, loop, break, ref, par y critical. Tratamos opt, alt, loop y break en detalle en los siguientes subapartados y tratamos ref en el siguiente capítulo. Los operadores par y critical van sobre concurrencia, que es un aspecto de diseño. Tratamos esto en el capítulo 20 cuando tratemos la concurrencia. Los otros operadores raramente se utilizan y dispone de suficiente información en la tabla para aplicarlos en caso de que los necesitara.

12.10.1. Ramificación con opt y alt

La figura 12.14 ilustra la sintaxis de opt y alt. El operador opt indica que su único operando se ejecuta si, y sólo si, la condición de protección es verdadera. De

lo contrario, la ejecución continúa después del fragmento combinado. `opt` es equivalente al constructor de programación:

```
if (condición1) then
    acción1
```

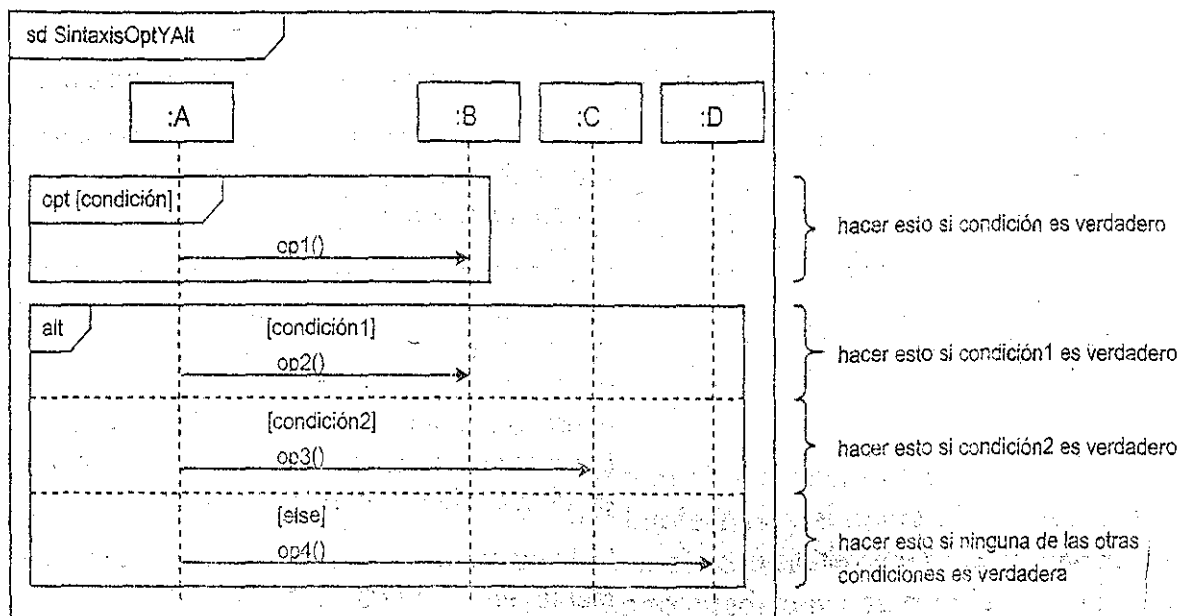


Figura 12.14.

El operador `alt` representa una opción entre alternativas. Cada uno de sus operandos tiene su propia condición de protección y solamente se ejecutará si la condición de protección es verdadera. Un operando opcional con una condición de protección de `[else]` se ejecuta si ninguna de las otras condiciones de protección son verdaderas.

Merece la pena indicar que solamente uno de los operandos `alt` se puede ejecutar. Esto significa que todas las condiciones de protección del operando deben ser exclusivas mutuamente. Si en cualquier momento, más de una condición de protección es verdadera, ésta es una condición de error y el comportamiento resultante de `alt` no está definido por la especificación de UML.

`alt` es equivalente al constructor de programación:

```
if (condición1) then
    operando 1
else if (condición2) then
    operando 2
...
else if (condiciónN) then
    operando N
else
    operando M
```

Puede ver que `opt` es semánticamente equivalente a un operador `alt` con exactamente uno operando. Como ejemplo de utilizar `opt` y `alt`, considere el caso de uso de la figura 12.15. Vimos por primera vez este caso de uso en el capítulo 4. Es

parte de un sencillo sistema de comercio electrónico y describe la actualización de la cantidad de un artículo en el carro de la compra de un cliente o eliminar el artículo por completo.

Caso de uso: GestionarCarro
ID: 2
Breve descripción: El Cliente cambia la cantidad de un artículo en el carro.
Actores principales: Cliente
Actores secundarios: Ninguno.
Precondiciones: 1. El contenido del carro de la compra es visible.
Flujo principal: 1. El caso de uso empieza cuando el Cliente selecciona un artículo en el carro. 2. Si el Cliente seleccionar "eliminar artículo" 2.1 El sistema elimina el artículo del carro. 3. Si el Cliente escribe una nueva cantidad 3.1 El sistema actualiza la cantidad del artículo en el carro.
Postcondiciones: Ninguna.
Flujos alternativos: Ninguno.

Figura 12.15.

La figura 12.16 es el diagrama de clase de análisis para este caso de uso. Puede ver que el CarroCompra alberga uno o más Artículos. Cada uno de estos Artículos es una cantidad de un Producto determinado. En los diagramas de clase de análisis, solamente muestra las clases, atributos y operaciones que ilustran el punto que está tratando. En este diagrama, estamos tratando de mostrar la colaboración entre CarroCompra, Artículos y Productos.

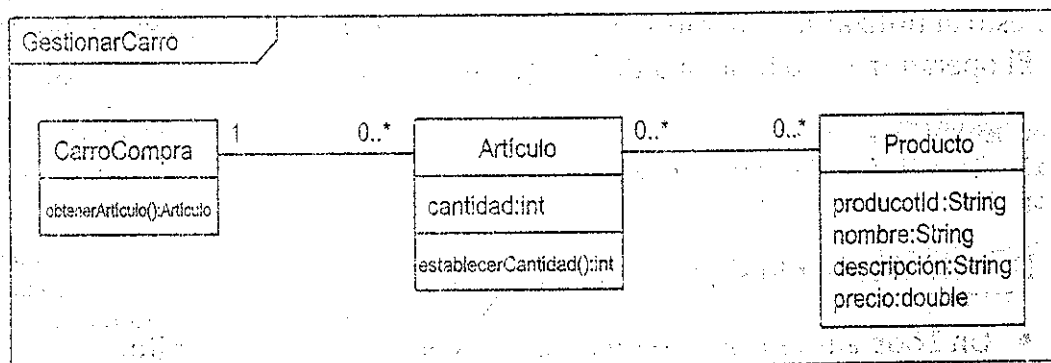


Figura 12.16.

Por último, la figura 12.17 muestra el diagrama de secuencia que realiza este caso de uso. Observe que en este diagrama de secuencia, hemos descrito el momen-

to en el que cuando la cantidad de un artículo cae a cero, se destruye. Esto es perfectamente razonable ya que un Artículo solamente existe para representar una cantidad de un Producto determinado en el CarroCompra. Sin embargo, podría argumentar que este nivel de detalle es innecesario en análisis y que un diagrama de secuencia que no muestra la eliminación explícita de Artículos sería igualmente aceptable.

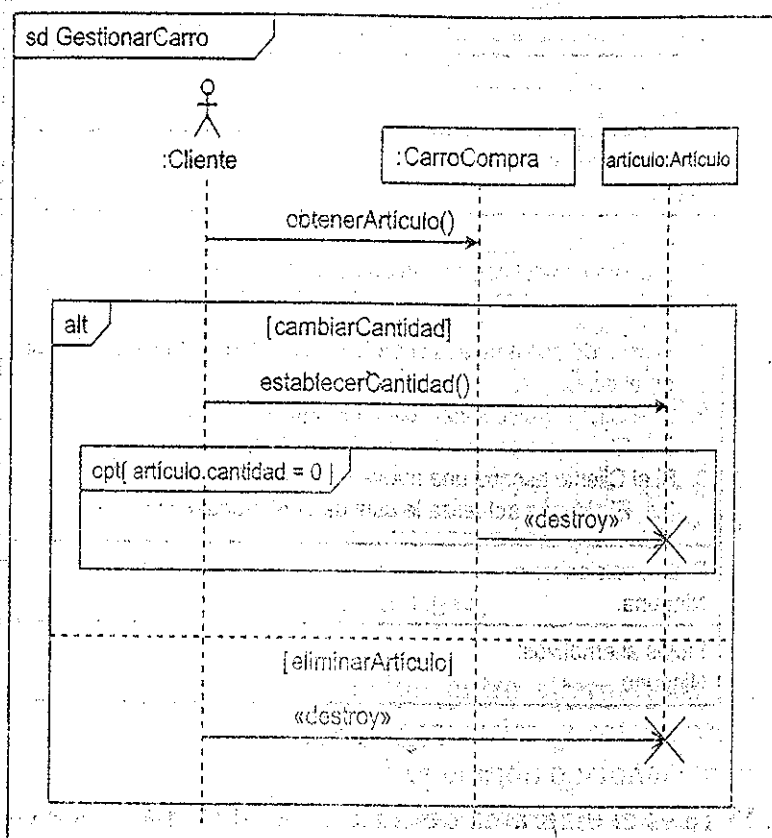


Figura 12.17.

12.10.2. Iteración con loop y break

Muy a menudo necesita mostrar bucles en diagramas de secuencia. Puede realizar esto al utilizar los operadores loop y break como se ilustra en la figura 12.18.

El operador loop funciona de la siguiente forma:

```

loop min veces then
while (condition is verdadera)
loop (max - min) veces

```

Debería observar lo siguiente sobre la sintaxis de loop:

- Un loop sin max, min o una condición es un bucle infinito.
- Si solamente se facilita min, entonces max=min.
- La condición es normalmente una expresión booleana pero también puede ser texto arbitrario.

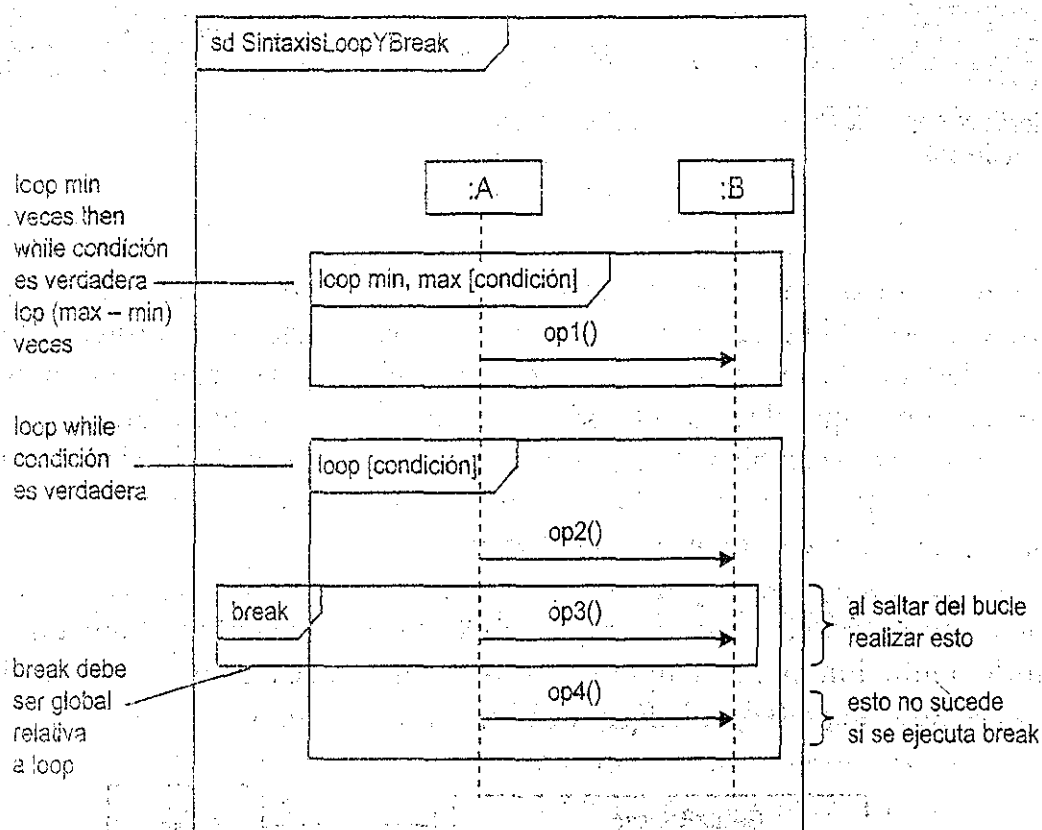


Figura 12.18.

Loop podría parecer bastante complejo al principio, pero se puede utilizar para dar soporte a una gran variedad de tipos de bucle. Algunos de los más comunes se listan en la tabla 12.4.

Tabla 12.4.

Tipo de bucle	Semántica	Expresión de bucle
while (true) {cuerpo}	Pasar en bucle sin fin.	loop o loop *
for i = n to m {cuerpo}	Repetir (m - n) veces.	loop n, m
while (expresiónBooleana) {cuerpo}	Repetir mientras expresiónBooleana es verdadera.	loop [expresiónBooleana]
repeat {cuerpo} while (expresiónBooleana)	Ejecutar una vez luego repetir mientras expresiónBooleana es verdadera.	loop 1, *[expresiónBooleana]
forEach objeto en colección {cuerpo}	Ejecutar el cuerpo del bucle una vez para cada objeto en una colección de objetos.	loop [para cada objeto en colecciónDeObjetos]

Tipo de bucle	Semántica	Esquema del bucle
forEach objeto de clase {cuerpo}	Ejecutar el cuerpo del bucle una vez para cada objeto de una clase determinada.	loop[para cada objeto en NombreClase]

Puede utilizar `break` para indicar bajo qué condiciones el bucle se rompe y qué pasa entonces. El operador `break` tiene una sola condición de protección y si ésta es verdadera, el cuerpo de `break` se ejecuta y el `loop` se termina. El punto fundamental aquí es que el resto del `loop` detrás del `break` no se ejecuta.

El fragmento combinado `break` está lógicamente fuera del bucle; no es parte de éste. Por lo tanto, siempre debe dibujar el fragmento `break` fuera de `loop`, pero solapándolo, como se muestra en la figura 12.18.

Uno de los tipos de bucle más comunes es atravesar una colección de objetos. Por ejemplo, puede utilizar un `loop` sobre una colección como una posible implementación para la operación `encontrarCurso(...)` de la clase `GestorRegistro` (véase la figura 12.19).

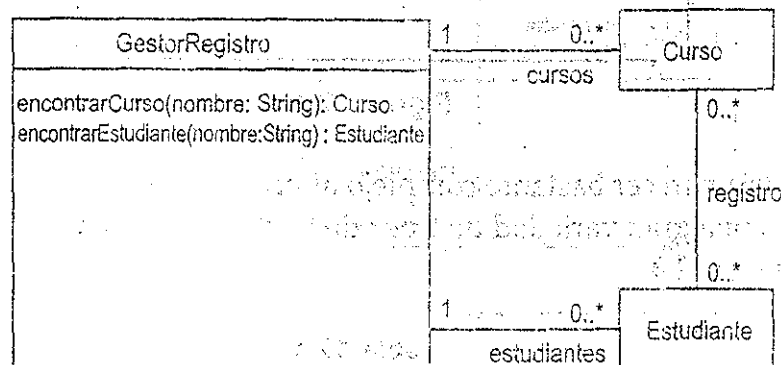


Figura 12.19.

La operación devuelve el objeto `Curso` con el nombre correcto o `null`, como se muestra en la figura 12.20. Puede utilizar el nombre de un extremo de la asociación con multiplicidad mayor que 1 como una colección de objetos. En este caso utilizamos `cursos` para representar una colección de objetos `Curso`.

12.11. Diagramas de comunicación

Los diagramas de comunicación enfatizan los aspectos estructurales de una interacción; cómo se conectan las líneas de vida. En UML 2, son semánticamente bastante débiles comparadas con los diagramas de secuencia. Ya ha visto algunos diagramas de comunicación. Según la especificación de UML 2.0 [UML2S], los diagramas de objetos que vio en el capítulo 7 se pueden considerar casos especiales de diagramas de clase o casos especiales de diagramas de comunicación de forma de instancia donde cada línea de vida representa una instancia de clase (objeto).

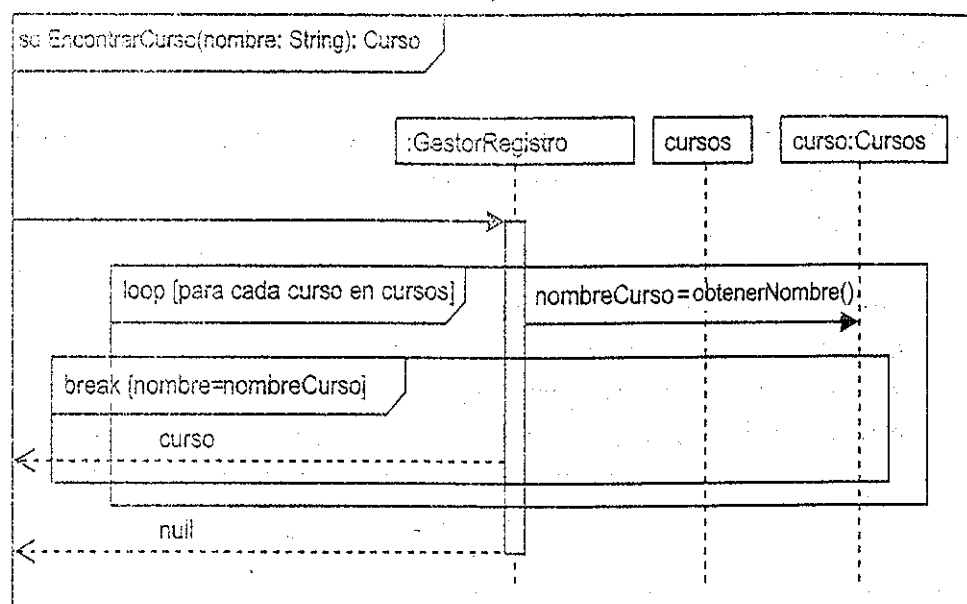


Figura 12.20.

Los diagramas de comunicación tienen una sintaxis similar a la de los diagramas de secuencia excepto que las líneas de vida no tienen líneas verticales discontinuas. En su lugar, se conectan mediante vínculos que proporcionan canales de comunicación para los mensajes a pasar. La secuenciación se indica al numerar cada mensaje jerárquicamente. La figura 12.21 muestra un sencillo diagrama de comunicación para el caso de uso AñadirCursos que muestra al :Secretario añadiendo dos nuevos cursos. Observe cómo los mensajes están numerados para indicar su secuencia y su anidamiento dentro de otros mensajes.

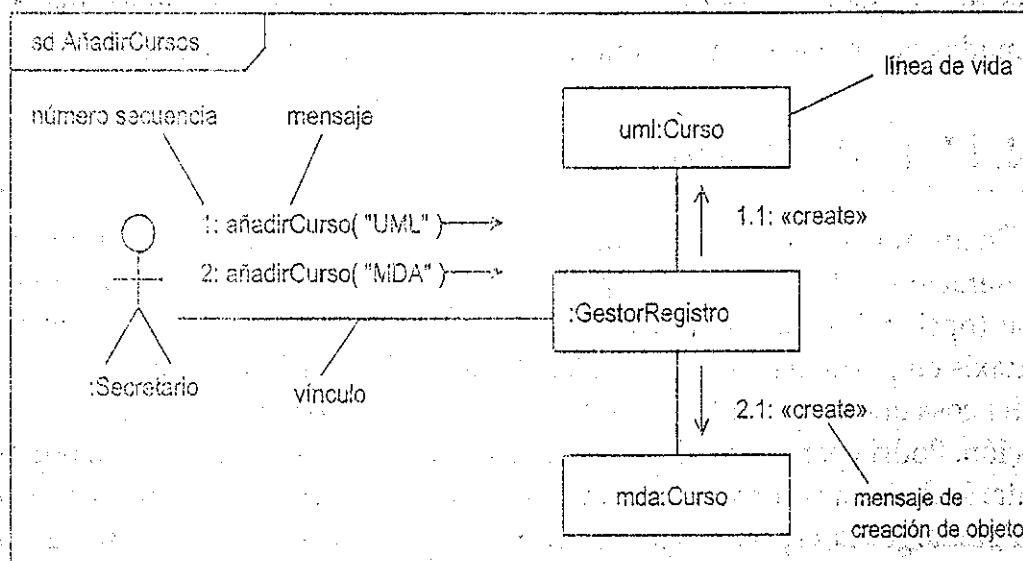


Figura 12.21.

Aquí está el itinerario para la figura 12.21.

- añadirCurso("UML"): El mensaje añadirCurso(...) se envía a la línea de vida :GestorRegistro con el parámetro "UML". La instancia

:GestorRegistro invoca una operación denominada `añadirCurso(...)` con el parámetro "UML" y el foco del control pasa a esta operación.

1.1. `<<create>>`: Puesto que el número de secuencia es 1.1, esto nos dice que nos encontramos dentro del foco de control de la operación `añadirCurso(...)`. :GestorRegistro envía un mensaje anónimo que se estereotipa `<<create>>`. Estos mensajes `<<create>>` crean nuevos objetos, y este mensaje en particular crea un nuevo objeto, `uml:Curso`. Asignará a este mensaje anónimo un nombre, y posiblemente parámetros, más adelante en análisis o diseño pero por ahora es suficiente con mostrar que está creando un nuevo objeto `uml:Curso`. Después de la creación del objeto, ya no se envían más mensajes dentro del foco de control de `añadirCurso(...)`, y este flujo regresa.

2. `añadirCurso("MDA")`: El mensaje `añadirCurso(...)` se envía a :GestorRegistro con el parámetro "MDA". El foco de control pasa a la operación `añadirCurso(...)`.

2.1. Puesto que la secuencia está numerada como 2.1, esto nos dice que estamos dentro del foco de control de `añadirCurso(...)`. El :GestorRegistro envía un mensaje anónimo estereotipado `<<create>>` que crea un nuevo objeto, `mda:Curso`. Después de la creación del objeto, el foco de control de `añadirCurso(...)` regresa y la interacción termina.

Al principio los diagramas de comunicación pueden ser algo complicados de leer ya que suceden muchas cosas. Los puntos clave a realizar son que un mensaje envía resultados en una operación que se invoca en una instancia y que la numeración de los mensajes indica el anidamiento de llamadas de operación dentro de llamadas de operación (por ejemplo, el foco anidado de control).

12.11.1. Iteración

Puede mostrar iteración en diagramas de comunicación al utilizar una expresión de iteración. Esto implica un especificador de iteración (*) y una cláusula de iteración (opcional) como se muestra en la figura 12.22. UML 2 no prescribe ninguna sintaxis en particular para cláusulas de iteración, por lo tanto puede utilizar cualquier cosa que tenga sentido. Por lo general, el código o pseudocódigo es una buena opción. Podría pensar que los diagramas de comunicación utilizarían por defecto la sintaxis de iteración para diagramas de secuencia que describimos anteriormente. Sin embargo, la especificación UML no lo menciona. Si decide utilizar la misma sintaxis de iteración, una expresión de iteración se podría escribir como:

```
*[loop min, max[condición]]
```

Esto tiene la ventaja de coherencia pero tiene cierta redundancia sintáctica ya que `loop` y `*` son especificadores de iteración. No obstante, consideramos que se trata de un buen enfoque.

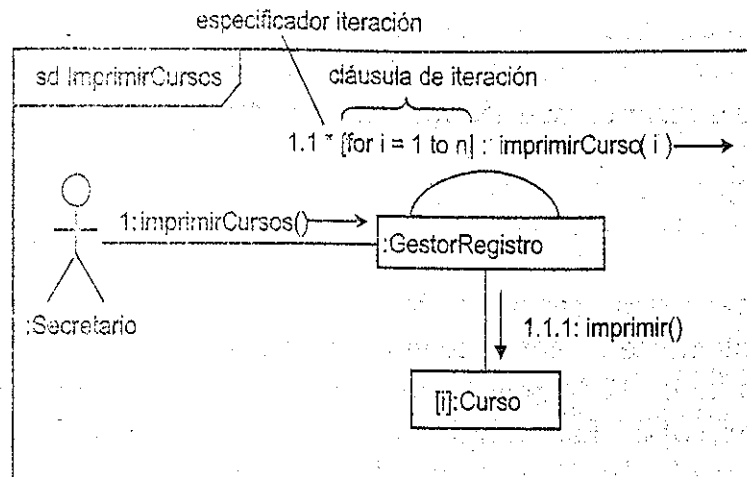


Figura 12.22.

En el ejemplo de la figura 12.22, hemos utilizado pseudocódigo para indicar que la cláusula de iteración pasa en bucle incrementando i de 1 a n . Luego utilizamos i como un selector para una instancia *Curso* específica a la que enviamos el mensaje `imprimir()`.

Esto tiene el efecto de imprimir todas las instancias *Curso*. Sin embargo, este enfoque asume que las instancias *Curso* están almacenadas en algún tipo de colección indexada. Si no quiere realizar esa asunción, puede utilizar el enfoque alternativo mostrado en la figura 12.23.

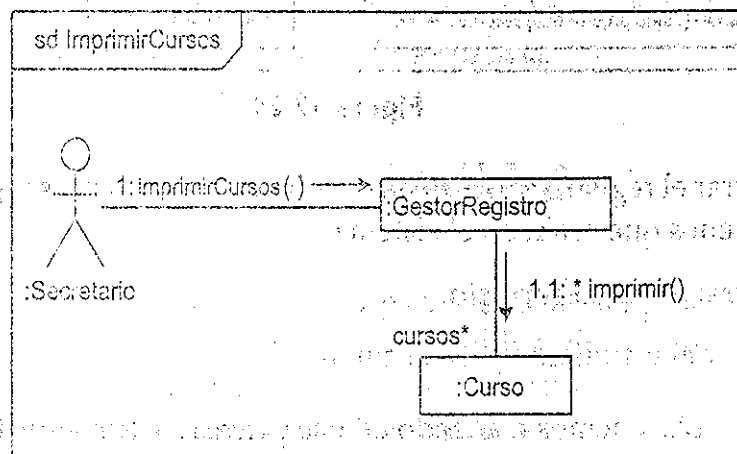


Figura 12.23.

En la figura 12.23 hemos realizado lo siguiente:

1. Hemos mostrado el nombre de rol y multiplicidad en `:GestorRegistro` hacia el vínculo `:Curso`. Esto indica que `:GestorRegistro` está conectado a una colección de objetos `:Curso` por medio del nombre de rol `cursos` (véase el diagrama de clase en la figura 12.7).
2. Hemos añadido el especificador de iteración al mensaje `imprimir()`. Esto indica que el mensaje `imprimir()` se envía a cada objeto en la colección.

El especificador estándar de iteración (*) significa que los mensajes se ejecutarán de modo secuencial. Si desea indicar que los mensajes se ejecutan todos en paralelo, debe utilizar el especificador de iteración paralelo *//.

12.11.2. Ramificación

Puede modelar ramificación al añadir condiciones de protección a los mensajes. El mensaje solamente se envía cuando la condición de protección evalúa como verdadero. La figura 12.24 muestra un ejemplo de ramificación en nuestro sistema de registro de curso. Este diagrama de comunicación realiza el caso de uso RegistrarEstudianteParaCurso. En este sistema de registro, el registro es un proceso de tres pasos:

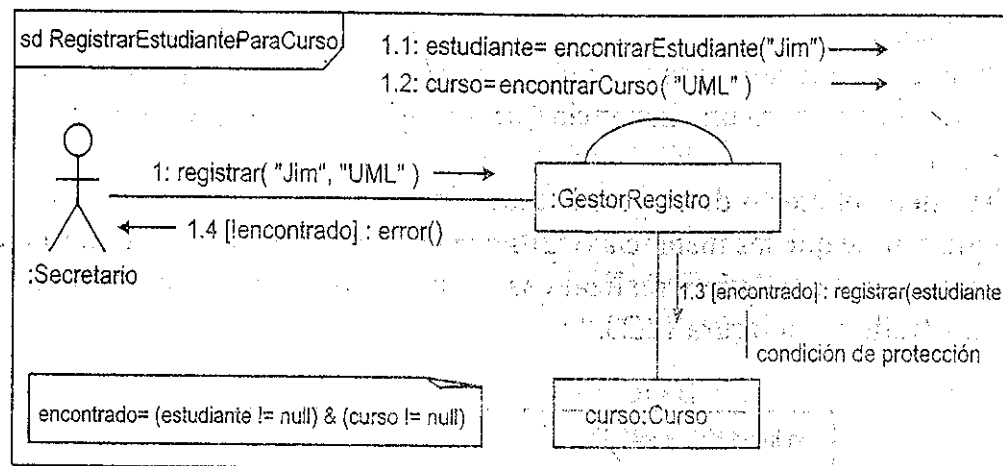


Figura 12.24.

- Encontrar el registro correcto de estudiante, no podemos registrar a estudiantes a menos que estén en el sistema.
- Encontrar el curso correcto.
- Registrar al estudiante para el curso.

En la figura 12.24 hemos realizado un uso extensivo de condiciones para mostrar cómo puede utilizarlas en diagramas de comunicación.

Las condiciones no tienen una sintaxis formal pero son expresiones que implican variables temporales en el ámbito de foco actual de control, o atributos de las clases implicadas en la iteración.

En la figura 12.24 registramos los resultados de las operaciones encontrarEstudiante(...) y encontrarCurso(...) en dos variables temporales: estudiante y curso. Luego utilizamos los valores de estas variables para calcular el valor de la variable temporal booleana encontrado. Utilizamos encontrado para crear una ramificación en el paso 1.3. También lo utilizamos para decidir si realizamos una condición de error para el :Secretario en el paso 1.4. Aquí tiene el itinerario para la figura 12.24.

1. registrarEstudiante("jim", "UML"): El actor :Secretario envía el mensaje registrarEstudiante("jim", "UML") a :GestorRegistro.
 - 1.1. encontrarEstudiante("jim"): El :GestorRegistro se envía el mensaje encontrarEstudiante("jim"). El valor de retorno de esta operación se almacena en la variable estudiante. Será null si la búsqueda falla.
 - 1.2. encontrarCurso("UML"): El :GestorRegistro se envía el mensaje encontrarCurso("UML"). El valor de retorno de esta operación se almacena en la variable curso. Será null si la búsqueda falla.
 - 1.3. [encontrado] registrar(estudiante): El :GestorRegistro envía el mensaje registrar(estudiante) al objeto curso. Este mensaje está protegido por una condición y solamente se enviará si estudiante y curso no son null. Es decir, solamente tratamos de registrar al estudiante con el curso si ambos objetos estudiante y curso se han encontrado con éxito.
 - 1.4. [!encontrado]:error(): Si encontrado es falso, invoque la operación error() en :Secretario.

Es bastante difícil mostrar la ramificación claramente en diagramas de comunicación, las condiciones parecen extenderse por todo el diagrama y puede hacerse complejo bastante rápido. Como una directriz general de estilo, solamente muestra una ramificación muy sencilla en estos diagramas. Es mucho más sencillo mostrar ramificaciones complejas en diagramas de secuencia.

12.12. ¿Qué hemos aprendido?

La realización de caso de uso es una parte esencial del proceso de análisis. Le permite comprobar sus teorías frente a la realidad al demostrar explícitamente cómo los objetos de sus clases pueden interactuar para proporcionar el comportamiento especificado del sistema. Los diagramas de iteración muestran cómo las clases y objetos realizan requisitos según se especifica en los casos de uso. Ha aprendido lo siguiente:

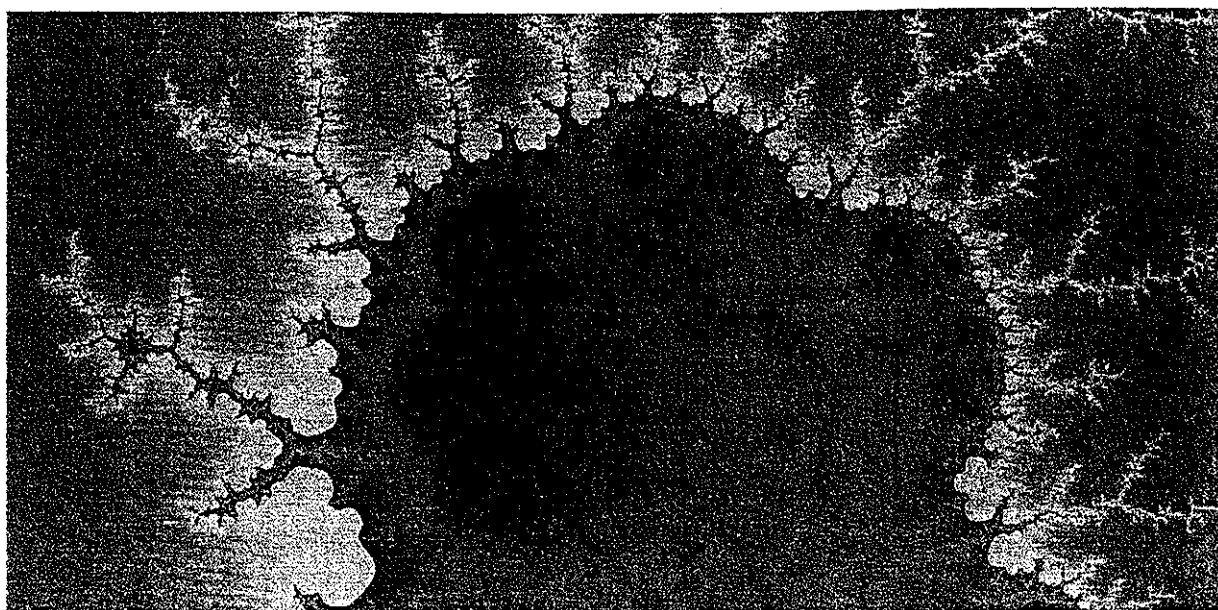
- La actividad UP Analizar un caso de uso es donde crea realizaciones de caso de uso; esta actividad crea parte de la vista dinámica del sistema.
- Las realizaciones de caso de uso muestran cómo las instancias de clase de análisis interactúan para realizar los requisitos funcionales especificados por un caso de uso.
 - Toda realización de caso de uso realiza exactamente un caso de uso.
 - Las realizaciones de caso de uso constan de:

- Diagramas de clase de análisis: Estos deberían "contar una historia" sobre uno (o más) casos de uso.
 - Diagramas de interacción: Estos demuestran cómo los objetos interactúan para realizar el comportamiento de caso de uso.
 - Requisitos especiales: Siempre descubre nuevos requisitos durante la realización de caso de uso y necesita grabarlos.
 - Mejora del caso de uso: Es posible que necesite cambiar un caso de uso cuando empiece a realizarlo.
- Las interacciones son unidades de comportamiento de un clasificador de contexto.
 - Las interacciones pueden utilizar cualquiera de las características del clasificador de contexto.
 - En la realización de caso de uso, el clasificador de contexto es un caso de uso.
 - Los diagramas de interacción de forma genérica muestran interacciones entre roles que las instancias de clasificador pueden desempeñar en la interacción.
 - Los diagramas de interacción de forma de instancia muestran interacciones entre instancias específicas de clasificador: utilice la notación normal de instancia para las líneas de vida.
 - Una línea de vida representa un participante en una interacción; cómo una instancia de un clasificador participa en la interacción.
 - Toda línea de vida tiene un nombre opcional, un tipo y un selector opcional.
 - Toda línea de vida se dibuja con el mismo icono que su tipo.
 - Subraye el nombre, tipo y selector para mostrar instancias reales.
 - Un mensaje representa un tipo específico de comunicación entre dos líneas de vida en una interacción.
 - Mensaje síncrono (punta de flecha completa).
 - Mensaje asíncrono (punta de flecha abierta).
 - Retorno de mensaje (punta de flecha abierta, línea discontinua).
 - Crear mensaje (punta de flecha abierta, línea continua, estereotipada <<create>>).
 - Destruir mensaje (punta de flecha abierta, línea continua, estereotipada <<destroy>>).
 - Mensaje encontrado (punta de flecha abierta, se origina de un círculo completo).

- Mensaje perdido (punta de flecha abierta, termina en un círculo completo).
- Diagramas de interacción:
 - Diagramas de secuencia: Enfatizan secuencia ordenada en el tiempo de envío de mensajes.
 - Diagramas de comunicación: Enfatizan relaciones estructurales entre objetos.
 - Diagramas de visión de interacción: Enfatizan relaciones entre interacciones.
 - Diagramas de tiempo: Enfatizan aspectos reales de interacciones.
- Diagramas de secuencia.
 - El tiempo corre de arriba abajo.
 - Las líneas de vida van de izquierda a derecha:
 - Las líneas de vida tienen líneas verticales discontinuas que indican la duración de la línea de vida.
 - Las líneas de vida pueden tener activaciones para indicar cuándo la línea de vida tiene foco de control.
Organice las líneas de vida para minimizar el número de líneas que se cruzan.
 - Sitúe scripts explicativos debajo de la parte izquierda del diagrama de secuencia.
 - Invariantes de estado. Sitúe símbolos de estado en la línea de tiempo en los puntos apropiados.
 - Restricciones: Sitúe restricciones en {} en o cerca de las líneas de vida.
- Fragmentos combinados: Áreas dentro de un diagrama de secuencia con comportamiento diferente.
 - El operador define cómo se ejecutan sus operandos.
 - La condición de protección define si su operando se ejecuta.
 - El operando contiene el comportamiento.
- Operadores.
 - opt: Existe un solo operando que se ejecuta si la condición es verdadera (como if...then).
 - alt: El operando cuya condición es verdadera se ejecuta.
 - loop: loop min, max [condición]:
 - loop o loop *: Pasa: en bucle indefinidamente.

- `loop n, m`: Pasar en bucle (m-n) veces.
- `loop [expresiónBooleana]`: Pasar en bucle mientras `expresiónBooleana` es verdadera.
- `loop 1, * [expresiónBooleana]`: Pasar en bucle una vez, luego pasar en bucle mientras `expresiónBooleana` es verdadera.
- `loop [para cada objeto en colecciónDeObjetos]`: Ejecuta el cuerpo del bucle una vez para cada objeto en la colección.
- `loop [para cada objeto en nombreClase]`: Ejecuta el cuerpo del bucle una vez para cada objeto de la clase.
- `break`: Si la condición de protección es verdadera, se ejecuta el operando, si no el resto de la interacción que engloba.
- `ref`: El fragmento combinado hace referencia a otra interacción.
- `par`: Todos los operandos se ejecutan en paralelo.
- `critical`: El operando se ejecuta sin interrupciones.
- `seq`: Los operandos se ejecutan en paralelo sujetos a la siguiente restricción: los eventos que lleguen en la misma línea de vida de diferentes operandos ocurren en la misma secuencia que ocurren los operandos.
- `strict`: Los operandos se ejecutan en secuencia estricta.
- `neg`: El operando muestra interacciones inválidas.
- `ignore`: Lista mensajes que se omiten intencionadamente de la interacción.
- `consider`: Lista mensajes que se incluyen intencionadamente en la interacción.
- `assert`: El operando es el único comportamiento válido en ese punto en la interacción.
- Diagramas de comunicación: Enfatizan los aspectos estructurales de una interacción:
 - Las líneas de vida están conectadas por vínculos.
 - Los mensajes tienen un número de secuencia; están numerados jerárquicamente según el anidamiento del foco de control.
- Iteración: Utilice un especificador de iteración (*) y una cláusula opcional de iteración en el mensaje.
 - La cláusula de iteración especifica el número de veces a pasar en bucle.
 - Puede utilizar lenguaje natural, pseudocódigo, código fuente o notación de bucle de diagrama de secuencia para la cláusula de iteración.

- Puede mostrar iteración sobre una colección de objetos al mostrar el nombre de rol y multiplicidad (>1) en el extremo destino del vínculo y prefijar el mensaje con el especificador de iteración (*). El mensaje se envía a cada objeto por turnos.
- Utilice el especificador paralelo de iteración `*//` para indicar que los mensajes se ejecutan en paralelo.
- Ramificación: Prefije mensajes con condiciones de protección. Los mensajes se ejecutan si la condición de protección es verdadera.
- Puede ser difícil mostrar la ramificación claramente en un diagrama de comunicación; para ramificaciones complejas, utilice los diagramas de secuencia.



13

Realización avanzada del caso de uso
