Unidad 3 / Escenario 6

Lectura fundamental

Diagramas de secuencia

Contenido

Palabras clave: análisis de *software*, diseño de *software*, diagramas de secuencia y diagramas de estado.

1. Diagramas de secuencia: acción en el tiempo

Hasta ahora se ha abordado dos grandes herramientas para definir y describir una solución de *software*: los casos de uso y el diagrama de clases. Sin embargo, cada una de ellas tiene ciertas deficiencias que no les permite, por separado, dar cuenta de la complejidad y forma real de acción del sistema. Mientras los casos de uso describen qué debe ser capaz de hacer el sistema y el diagrama de clases despliega la estructura estática con la que debe ser posible cumplir con los requerimientos, falta un elemento que describa de una manera real y operativa cómo se van a llevar a cabo los procesos para que el sistema pueda cumplir con su trabajo. Es allí donde entra el diagrama de secuencia. Los diagramas de secuencia son un tipo particular e importante de los diagramas UML de interacción. Estos últimos modelan las relaciones e interacciones que se dan entre las partes del sistema en tiempo de ejecución. Los diagramas de secuencia tienen su interés principal en documentar el orden de las interacciones entre las partes del sistema. Incluyen otra información de importancia también, pero su foco principal es ser una representación sencilla de las interacciones y el orden de las mismas.

1.1. Elementos en un diagrama de secuencia

Un diagrama de secuencia está conformado de manera principal por un conjunto de participantes que son quienes guían las interacciones. Los participantes se ubican de manera horizontal en el diagrama, como se muestra en la figura. Ya que el diagrama modela las interacciones en tiempo de ejecución, cada participante representa en realidad un objeto de una clase particular y esto se refleja en el nombre dado a cada uno: en primer lugar, se encuentra el nombre del objeto y separado de él por (:) se encuentra la clase a la que corresponde.

Figura 1. Participantes

Fuente: elaboración propia

Cada participante tiene una línea punteada que corre verticalmente desde su centro. Estas líneas son llamadas líneas de vida e indican que un participante existe y está listo para recibir mensajes invocaciones de métodos) de otros participantes en la interacción.

Ya que se ha mencionado que uno de los propósitos principales de un diagrama de secuencia es la representación del orden de las interacciones, resulta evidente que el sentido del tiempo juega un papel importante en un diagrama de este tipo. El tiempo en un diagrama de secuencia comienza en la parte superior de la página, justo debajo del participante que se encuentra más arriba, y aumenta a medida que se baja por el diagrama. El orden en que se ubican las interacciones en la página indica el orden en que suceden en tiempo de ejecución. La siguiente Figura 2 Línea de tiempo ilustra el concepto:

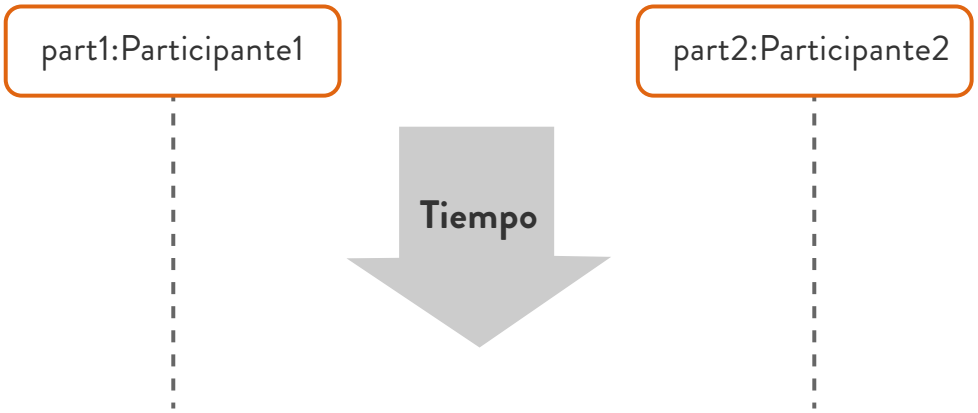

Figura 2. Línea de tiempo

Fuente: elaboración propia

En un diagrama de secuencia, la duración de las interacciones no es tan importante como el orden de estas. El orden es indicado por la posición de la interacción en el diagrama, pero cuánto espacio ocupe verticalmente una interacción dada no es indicativo de la duración de esta.

Con el claro ordenamiento de las interacciones se puede continuar con el próximo elemento clave de un diagrama de secuencia: los mensajes entre participantes. Una interacción sucede cuando un participante decide enviar a otro un mensaje.

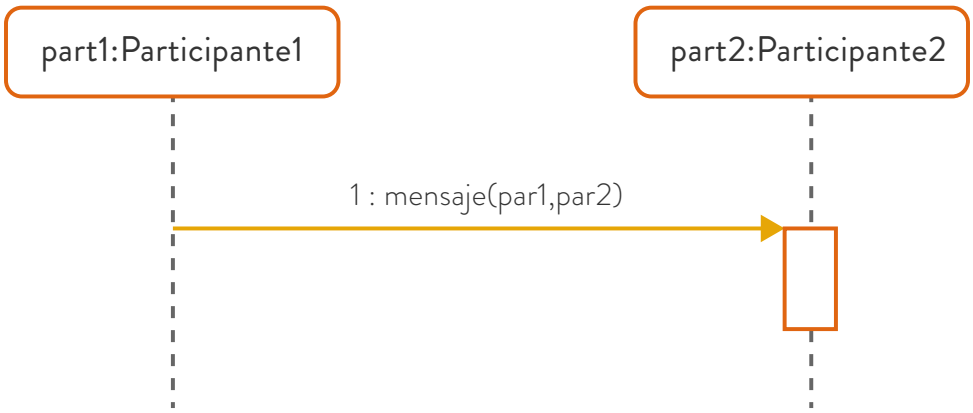

Figura 3. Mensajes entre participantes

Fuente: elaboración propia

Aquí puede verse la situación cuando el participante part1, de la clase Participante1 envía un mensaje al participante part2, de la clase Participante2. Inspeccionando la figura, es posible ver que aparecen nuevos elementos:

- El mensaje mismo es representado por una flecha en la dirección en que es enviado, es decir, yendo del origen del mensaje (en este caso part1), hacia el receptor del mismo (part2). Obsérvese la punta de la flecha. El tipo de punta será importante para definir qué tipo de mensaje ha sido enviado. Se revisará esta cuestión en más detalle más adelante.
- Sobre el mensaje va un texto que es conocido como la firma del mismo. En este texto se especifica qué mensaje ha sido enviado. Como se dijo con anterioridad, la mayor parte de los mensajes se traducen en la invocación de métodos pertenecientes a la clase que los recibe, de manera que el texto del mensaje es el estereotipo del método invocado, incluyendo los parámetros enviados para dicha invocación.
- Al principio de la firma del mensaje se encuentra un número. Este número identifica el orden del mensaje dentro del diagrama de secuencia, y es opcional. Resulta útil para diagramas particularmente complicados o cuando se desea hacer observaciones escritas acerca del diagrama a otra persona (Resulta mucho más sencillo hablar del mensaje número 7, que de la decimoséptima invocación del método calcular porcentaje.
- En la línea de vida del receptor del mensaje se crea un rectángulo. Este rectángulo indica que el receptor no solo existe, sino que se encuentra activo haciendo lo solicitado por el mensaje recibido. Este rectángulo se conoce como una barra de activación y es útil para identificar los periodos de actividad reales de un objeto a lo largo de una interacción.

Adicionalmente, y dependiendo del tipo de mensaje (ver numeral 1.2 Tipos de mensajes) un mensaje puede generar una respuesta por parte del receptor (se puede pensar en esto en términos del valor retornado luego de invocar un método, o simplemente el retorno de control del punto de ejecución cuando el método es de tipo *void*). La respuesta a un mensaje es representada mediante un tipo particular de flecha; una flecha punteada con la punta hueca representa el retorno de un mensaje previamente enviado:

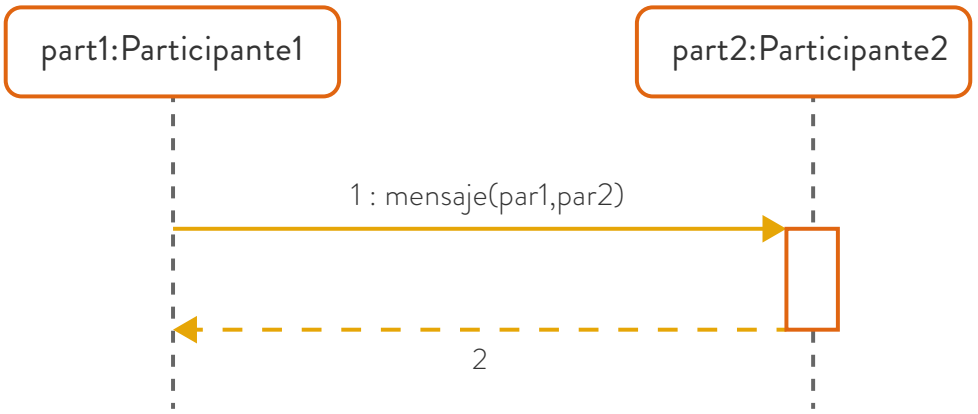

Figura 4. Tipos de mensajes

Fuente: elaboración propia

Ahora bien, las interacciones entre participantes pueden ser, y usualmente son, más complejas que un simple envío/recepción de mensajes uno a la vez. En la mayor parte de los casos, un mensaje recibido por un participante causa que este envíe a su vez mensajes a otros participantes, o que incluso invoque mensajes sobre sí mismo (ejecutando métodos que le pertenecen). Un ejemplo de estas situaciones es:

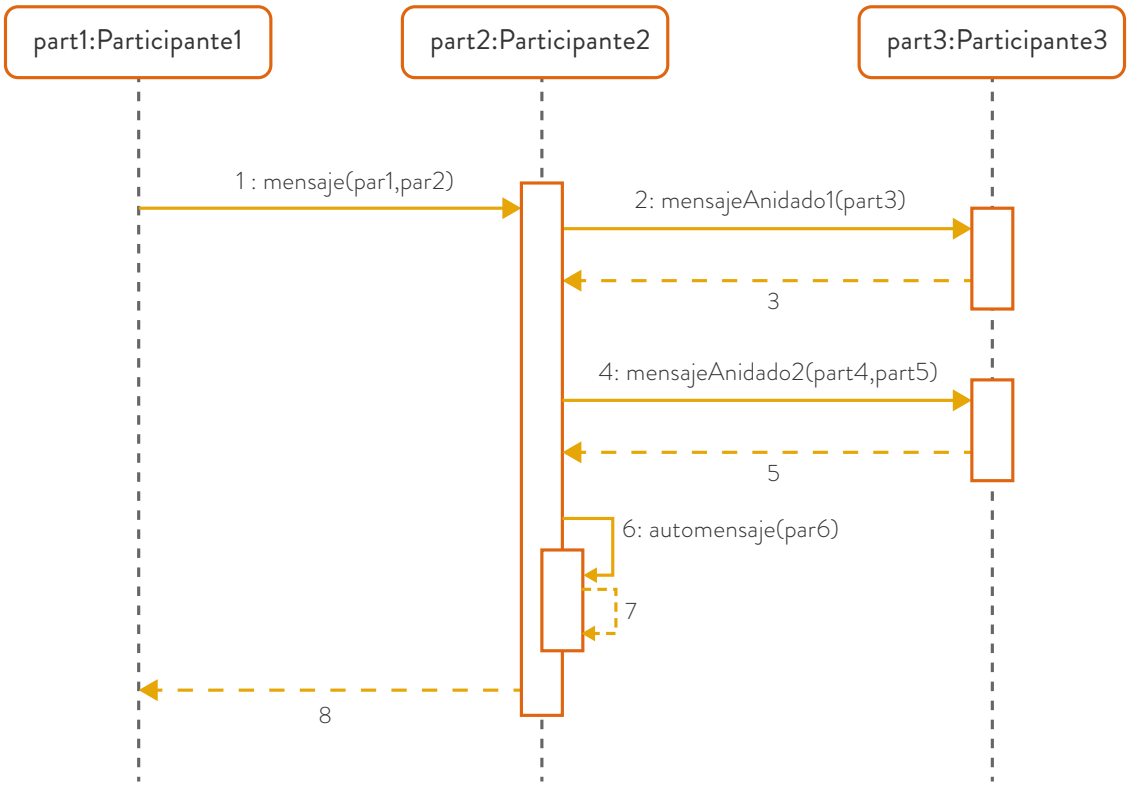

Figura 5. Mensaje sobre sí mismo

Fuente: elaboración propia

En este caso, el envío del mensaje inicial desde part1 hacia part2 produjo que part2 enviara dos mensajes (mensajeAnidado1 y mensajeAnidado2) hacia part3, recibiendo respuesta para cada uno de ellos, y luego, invocara un mensaje sobre sí mismo (auto Mensaje, que representa una llamada al método del mismo nombre de la clase Participante 2). Todo esto sucede antes de que el mensaje inicial, mensaje genere una respuesta para part 1. Esto resulta evidente si se leen los números que indican el orden de las interacciones, y además se apoya en el recurso visual mencionado al principio de la lectura, referente a que el orden de ubicación de los mensajes en el diagrama representa el orden de su ocurrencia en el tiempo. Todos los mensajes ubicados visualmente entre el mensaje 1 y el mensaje 8 serán ejecutados en el tiempo de manera similar entre el envío del mensaje 1 y la recepción de la respuesta correspondiente, en el mensaje 8. Como anotación final al tema de los mensajes anidados, es importante anotar que no existe límite para el número de mensajes anidados dentro de un mensaje particular, así como tampoco lo existe para el número de niveles de la invocación. En este caso, part3 pudo haber enviado un mensaje a otros participantes antes de retornar las respuestas a part2.

1.2. Tipos de mensajes

Con la estructura de un diagrama de secuencia descrita como se ha hecho hasta ahora es posible abordar un tema importante: los diferentes tipos de mensajes. Existen cinco tipos de mensajes básicos en un diagrama de secuencia:

Mensajes sincrónicos: representan la situación en la que el participante que envía el mensaje requiere de una respuesta desde el receptor del mismo antes de poder continuar con su ejecución y el proceso de envío de mensajes. Es el tipo más común de mensaje y, en términos de programación, equivale a la invocación normal de un método perteneciente al objeto receptor. Se representa con una flecha que tiene la punta rellena:

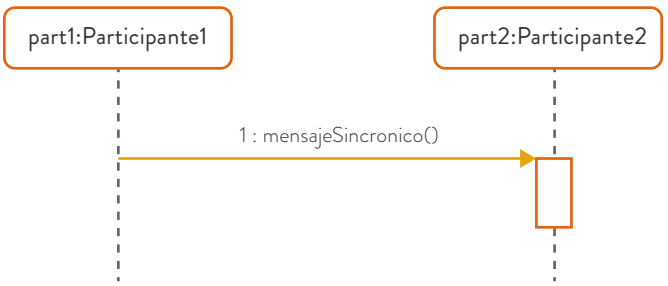

Figura 6. Mensaje sincrónico

Fuente: elaboración propia

Mensajes de retorno: cuando un participante termina la ejecución de lo solicitado por un mensaje recibido, puede retornar un valor al participante que envió el mensaje, o simplemente puede retornar el control del flujo de ejecución al dicho participante. En cualquier caso, este hecho es representado a través de un mensaje de retorno, denotado visualmente utilizando una flecha punteada con la punta hueca.

Figura 7. Mensaje de retorno

Fuente: elaboración propia

En este caso, la ejecución del mensaje sincrónico generó una respuesta desde part2 que vuelve a part1 bajo la forma del valor almacenado en valor respuesta. Algunas aplicaciones de diagramación UML y diseñadores de *software* consideran innecesaria la inclusión de mensajes de retorno en un diagrama de secuencia. A pesar de que según el estándar UML son en efecto opcionales, al igual que los números que indican el orden de los mensajes, las flechas de retorno pueden ser de gran utilidad cuando se trata de leer un diagrama de interacción complejo. Como ejercicio, se insta al lector a tratar de interpretar el diagrama de secuencia presentado con anterioridad como ejemplo de los mensajes anidados, sólo que esta vez sin contar con la inclusión de los mensajes de respuesta:

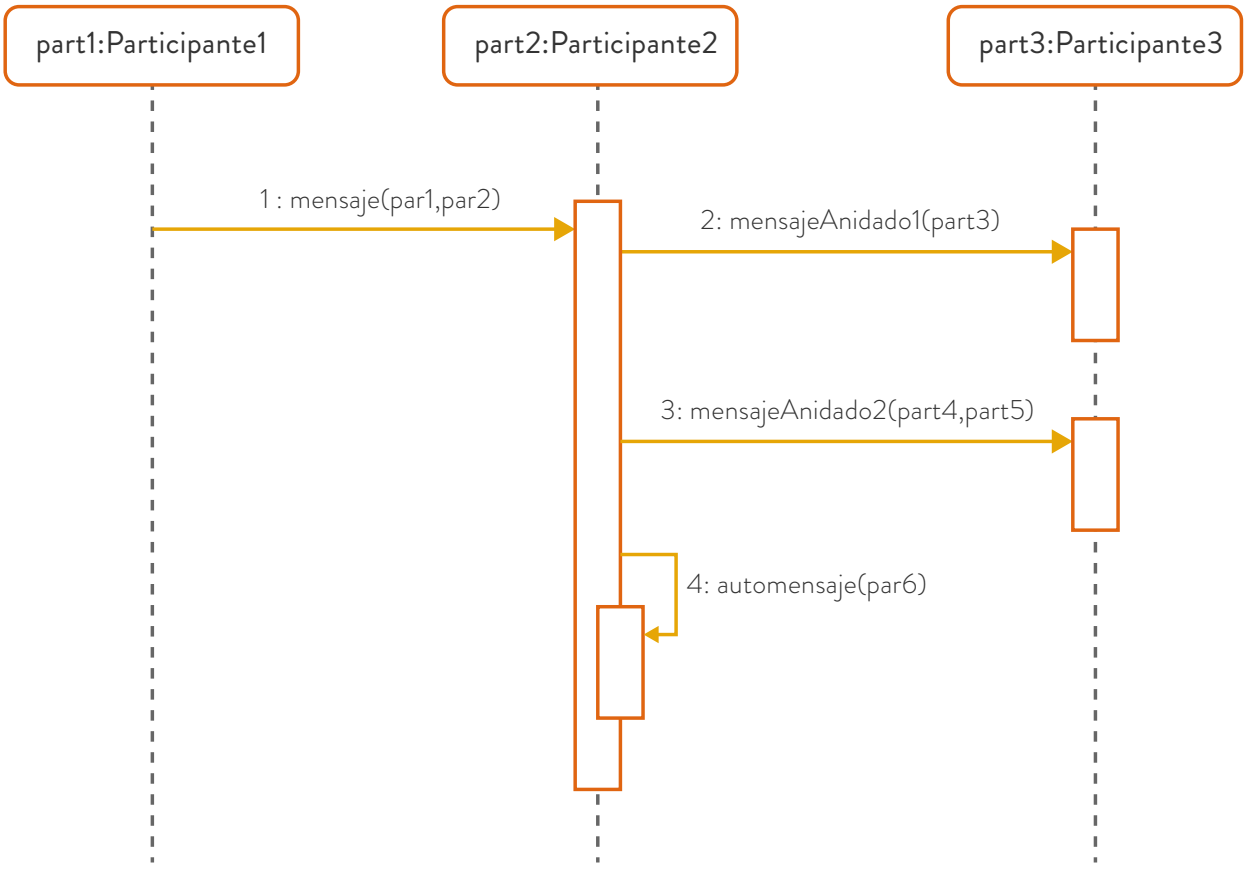

Figura 8. Mensajes de respuesta y anidada

Fuente: elaboración propia

Resulta evidente aquí la utilidad de las flechas que representan los mensajes de respuesta. Sin ellas no es claro si los mensajes 2, 3 y 4 se envían luego de que el mensaje 1 ha recibido respuesta, o si se encuentran anidados dentro de éste. Problemas de lectura de esta índole son mucho más probables y mucho más costosos en diagramas más complicados y, además de indeseables, son muy fáciles de evitar si se incluye de manera formal las flechas que indican los mensajes de retorno dentro del diagrama.

Mensajes asincrónicos: sirven para representar la situación en la que un mensaje es enviado, pero el participante que lo envía no necesita esperar por una respuesta antes de continuar con su ejecución y el envío de nuevos mensajes. En términos de implementación, los mensajes asincrónicos equivalen a la creación y lanzamiento de hilos de ejecución, que se desprenden de la línea principal de ejecución del programa y se convierten en líneas independientes que no dependen de la principal. Un mensaje asincrónico se representa mediante una flecha con la punta hueca:

Figura 9. Mensajes asincrónicos
Fuente: elaboración propia

Nótese que un mensaje asincrónico no tiene asociado un mensaje de respuesta, ni implica la creación de una barra de activación en el participante que lo recibe, por cuanto no es posible saber la duración del proceso que dispara el mensaje.

Mensajes de creación: un participante no tiene por qué existir a lo largo de toda la interacción. Puede ser creado solamente cuando es necesario. La creación de un participante (equivalente a la invocación del constructor de la clase correspondiente) se representa de una manera particular:

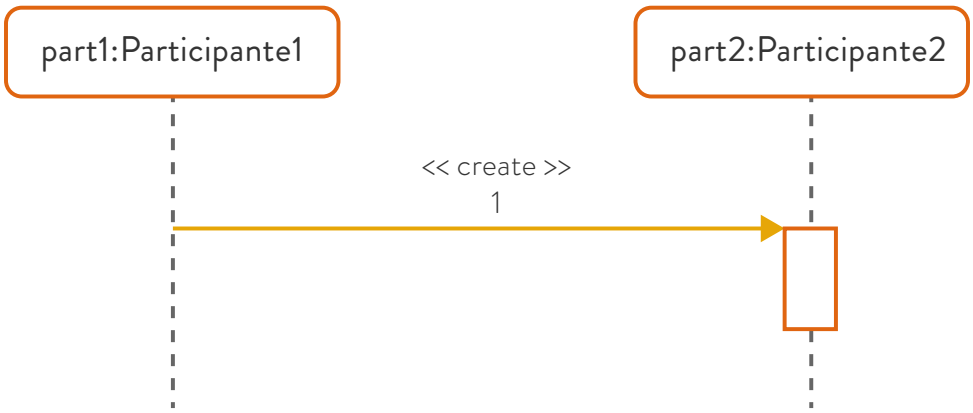

Figura 10. Mensaje de creación

Fuente: elaboración propia

En ocasiones, cuando el participante part1 requiere el objeto creado para su uso, es necesario representar de manera explícita el retorno del constructor:

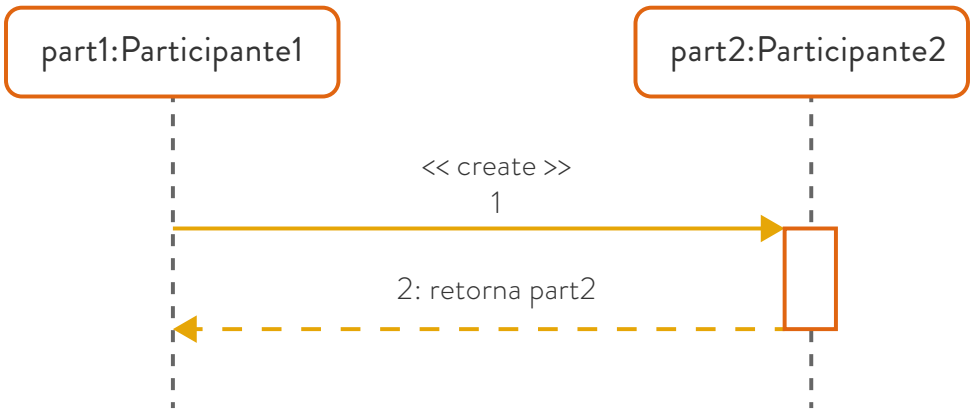

Figura 11. Mensaje de retorno del constructor

Fuente: elaboración propia

Nótese que el valor de retorno es el objeto creado.

Mensajes de destrucción: así como un objeto puede crearse en algún punto de una interacción, también puede ser destruido de ser necesario. Tal como sucede con un mensaje de creación, un mensaje de destrucción tiene una representación particular:

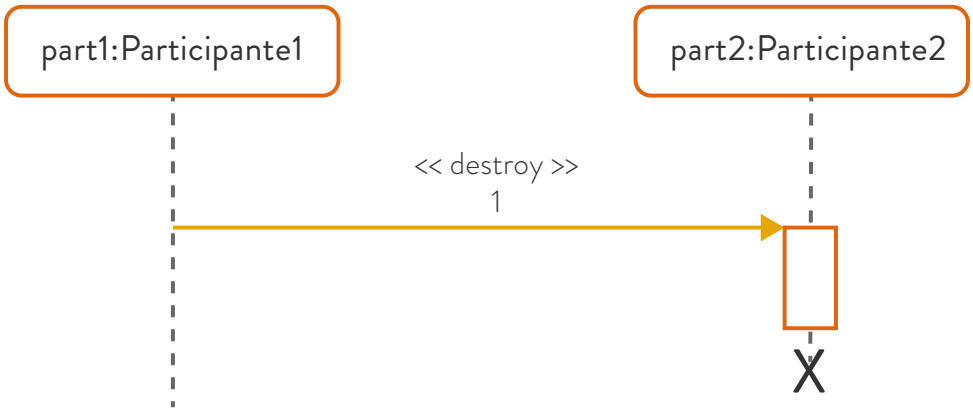

Figura 12. Mensaje de destrucción

Fuente: elaboración propia

Como elemento interesante, es de notar que la línea de vida del participante destruido es cortada, y una X es colocada al final de la misma para indicar su finalización.

1.3. Fragmentos

A estas alturas, y con los conceptos vistos, resulta evidente que modelar una interacción relativamente compleja resulta sencillo en términos operativos, pero puede producir un resultado difícil de leer por cuanto el diagrama de secuencia resultante puede ser bastante largo. Para facilitar la labor de organización, así como para permitir la inclusión de condicionales y ciclos dentro de la realización de un diagrama de secuencia, se introduce aquí el concepto de fragmento.

Un fragmento de un diagrama de secuencia es una caja que encierra un número de los mensajes e interacciones del diagrama. Un fragmento se superpone al diagrama y puede contener tantas interacciones como sean necesarias e incluso otros fragmentos, si el diagrama así lo requiere. Un fragmento tiene en la parte superior izquierda un operador, que indica su tipo:

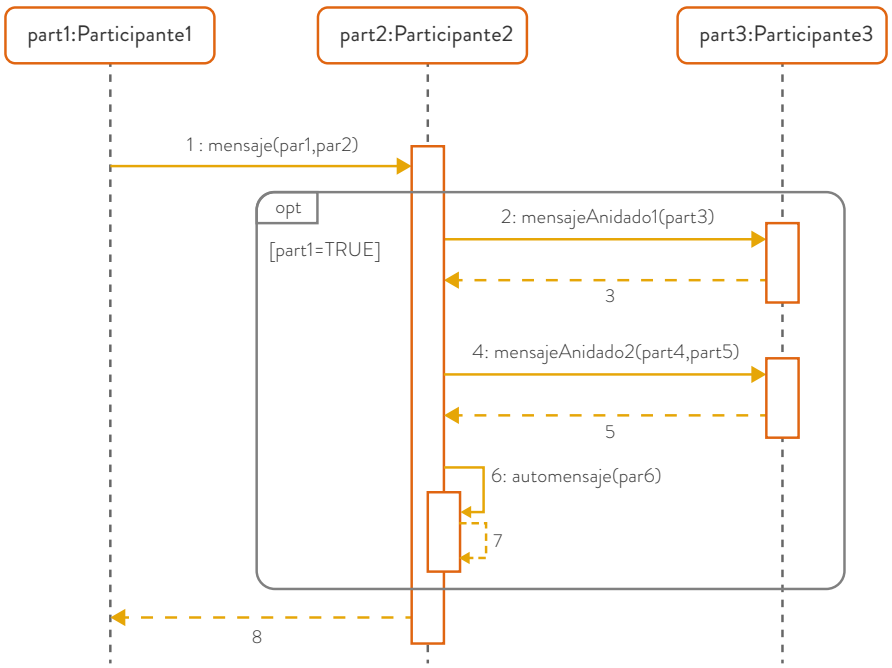

Figura 13. Fragmento opt.

Fuente: elaboración propia

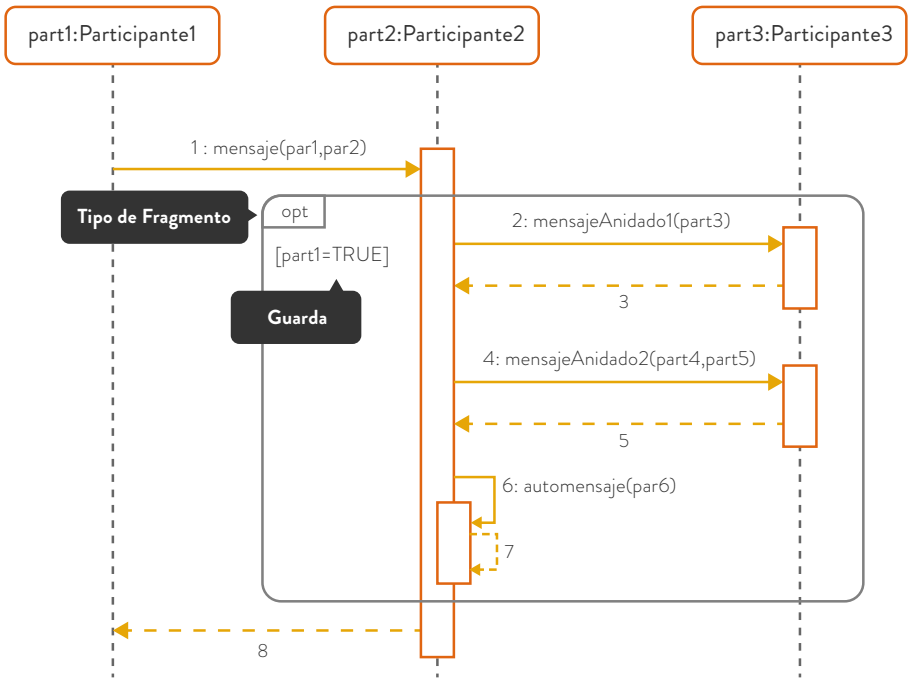

Figura 14. Fragmento opt. con guarda

Fuente: elaboración propia

En este caso se utilizó un fragmento de tipo *opt*, que indica que los mensajes contenidos dentro del fragmento serán ejecutados de manera opcional, dependiendo del resultado de la evaluación de la Guarda. La guarda es una condición de tipo booleano que determina si se ejecutan o no las interacciones contenidas en el fragmento, de manera muy similar a como lo hace una instrucción condicional en un programa. De hecho, si se desea asociar el fragmento opt con una estructura de control sería muy similar a una instrucción if, y la guarda se convertiría entonces en la condición a evaluar para determinar si se entra o no al if, para ejecutar las instrucciones allí contenidas.

Fragmento *alt*. En ocasiones, una sola condición no es suficiente para indicar el flujo de una interacción. En esas situaciones, un fragmento de tipo *alt* es la solución.

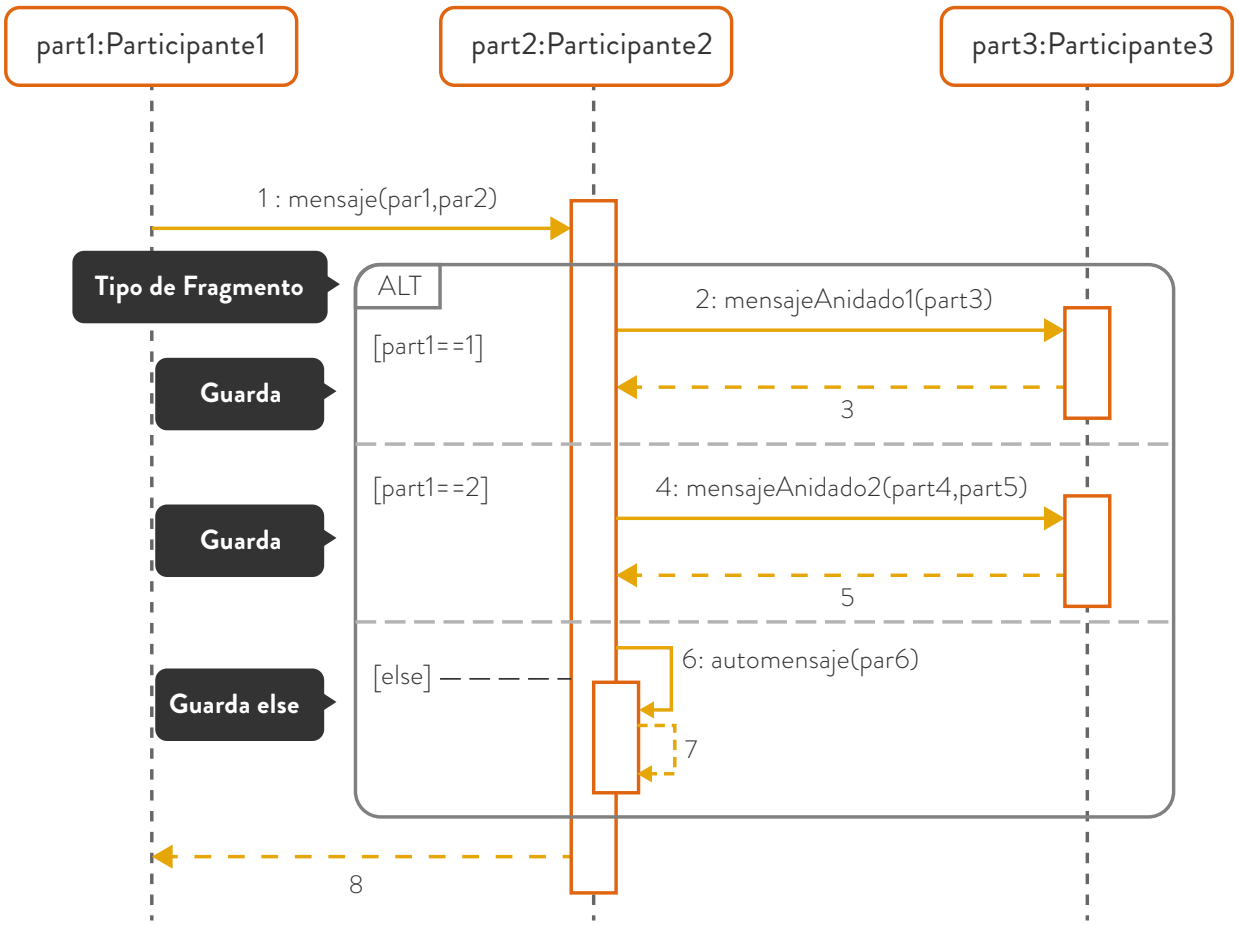

Figura 15. Fragmento *alt*

Fuente: elaboración propia

Un fragmento alt tiene múltiples guardas que indican múltiples puntos de entrada a la ejecución de las interacciones contenidas en el fragmento principal. Si se quiere establecer una relación de similitud con alguna estructura de control, un fragmento ***alt*** correspondería a una sentencia de tipo ***switch-case.*** Es acostumbrado incluir al final una guarda de tipo [***else***] para que se ejecuten las interacciones allí contenidas en caso de que ninguna de las guardas previas haya sido disparada.

Fragmento loop. Otro tipo de fragmento que resulta altamente útil es el fragmento ***loop***. Este tipo de fragmento indica que las interacciones en él contenidas deben ser ejecutadas tantas veces como indique la guarda definida. Un ejemplo de un fragmento de tipo ***loop*** es:

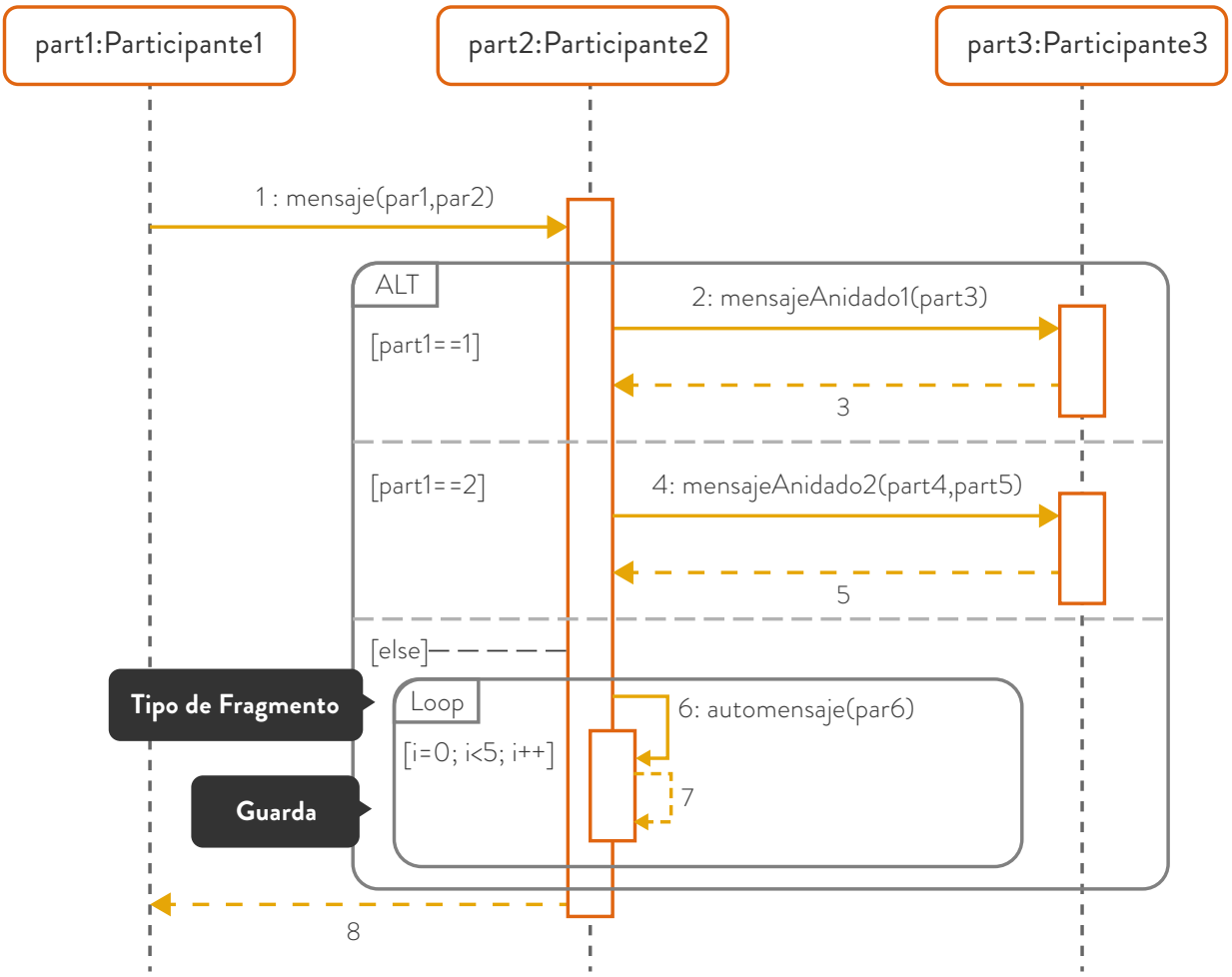

Figura 16. Fragmento *loop*

Fuente: elaboración propia

Nótese que, en este caso, el fragmento ***loop*** está inmerso dentro de la opción ***else*** del fragmento ***alt***. En este caso, la guarda definida indica que las interacciones contenidas en el fragmento deben ejecutarse cinco veces; la notación utilizada es semejante a la usada cuando se define una instrucción de control de tipo ***for***. De hecho, el comportamiento descrito por un fragmento ***loop*** es equivalente al descrito por una estructura de control ***for***.

Referencias

Fowler, M. (1999). *UML gota a gota*. México, D. F, México: Addison Wesley

INFORMACIÓN TÉCNICA

FACULTAD DE
INGENIERÍA, DISEÑO
E INNOVACIÓN

Módulo: Ingeniería del *Software* I

Unidad 3: Metodologías de desarrollo de *software*

Escenario 6: Modelamiento de *software*

Autor: Diana Angélica Cruz Ortega

Asesor Pedagógico: Luisa Esperanza Rincón Jiménez

Diseñador Gráfico: Cristian Navarro

Asistente: Ginna Quiroga