UT03 -Metodología OO - UML - Diagramas (parte 2)

ÍNDICE

Índice de contenido

Versión del documento

04/04/12 Creación del documento y diagramas de secuencias
13/04/12 Diagramas de Colaboraciones
06/05/12 Teoría últimos diagramas

1 DIAGRAMAS de SECUENCIAS

Muestra la forma en que los objetos se comunican entre sí con el paso del tiempo. Los diagramas de estado se centraban en un solo objeto, mostrando los cambios por los que pasaba dicho objeto.

Pero UML permite ver cómo cada objeto interacciona con el resto. Esta interacción se produce a lo largo del tiempo, siguiendo una secuencia que se toma un tiempo en llegar desde el principio hasta el fin.

En el diagrama de secuencias, los objetos se representan como habitualmente, mediante rectángulos. Como lo que se representa es **la instancia del objeto**, el nombre del objeto se subraya.

Se dibuja cada objeto en una columna de tiempo, en vertical, que representa su "**línea de vida**".

Cada objeto se representa arriba, justo en el comienzo de su línea de vida. En la línea de vida de cada objeto se colocan unos rectángulos llamados **activaciones**. Cada una de esas activaciones representa la ejecución de una operación (método) de ese objeto.

Los mensajes entre objetos (invocaciones de métodos de otro objeto desde uno dado) se representan mediante flechas orientadas.

1.1 Tipos de mensajes

Los mensajes que se envían son de tres tipos: simples, síncronos o asíncronos.

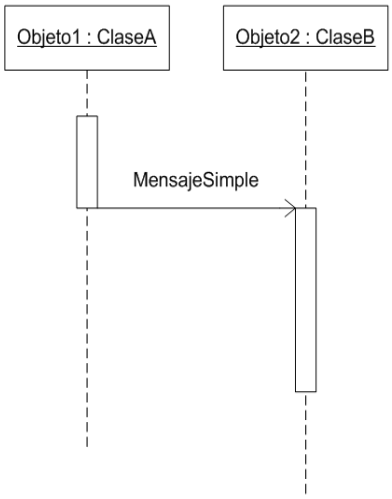

Ilustración 1: Mensaje simple

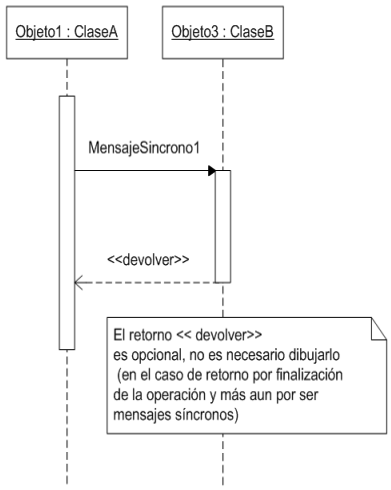

Ilustración 2: Mensaje síncrono

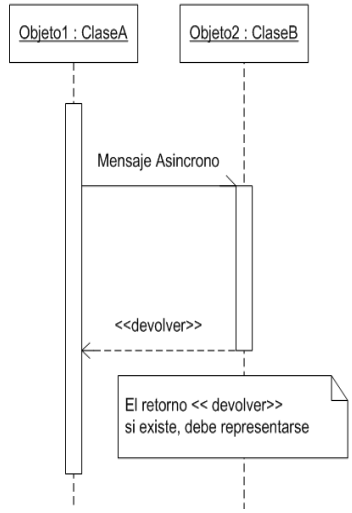

Ilustración 3: Mensaje asíncrono

El mensaje **simple**, solamente transfiere el control a otro objeto. Se representa con una punta de flecha como indica la Ilustración 1.

El **síncrono**, transfiere el control a otro objeto del que espera una respuesta. Mientras tanto, el objeto llamante queda bloqueado en espera de la respuesta. En este caso, no hay obligación de representar el retorno desde el objeto llamado. Tampoco es obligatorio en el caso de retornos por finalización de una operación. Se representa con una punta de flecha rellena.

El **asíncrono**, transfiere el control a otro objeto del que espera una respuesta. Pero en este caso, ambos objetos continúan ejecutándose simultáneamente. Por tanto, el objeto llamante NO queda bloqueado en espera de la respuesta. En este caso, hay que representar el retorno desde el objeto llamado, siempre que exista ese retorno. Se representa con media una punta de flecha, como en la Ilustración 3.

Se suele dibujar al actor que inicia el proceso sobre o al lado del primer objeto ejecutado.

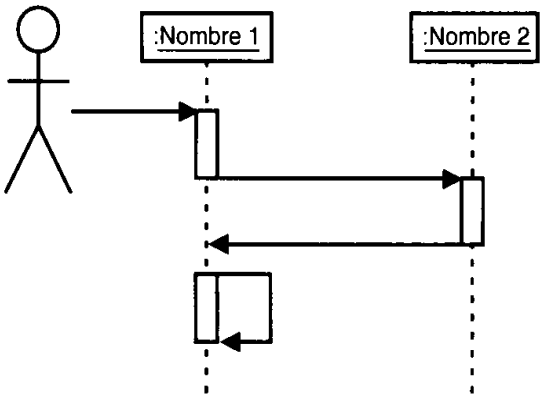

Ilustración 4: Actor que inicia

El retorno o devolución de un mensaje se representa como una flecha discontinua y suele llevar un estereotipo que indica que se trata de una devolución.

Además existen mensajes de creación de objetos, cuando el mensaje origina la creación de una nueva instancia de un objeto y de destrucción en el caso contrario.

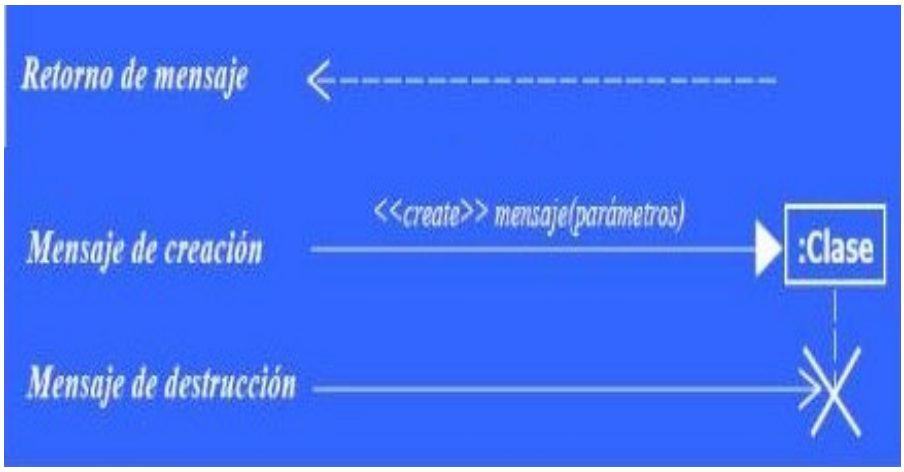

Ilustración 5: Mensajes

Un mensaje puede ser reflexivo si el mensaje se envía al mismo objeto. Si se invoca una nueva operación propia o la misma, se dibuja una nueva activación sobre la anterior. (En el segundo caso, se tratará de una llamada recursiva).

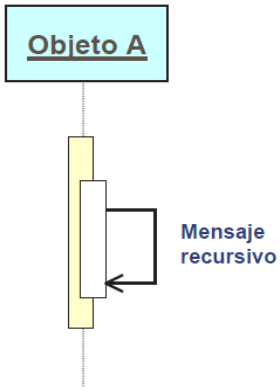

Ilustración 6: Mensaje Recursivo

1.2 Diagramas de secuencias de instancias y genéricos

Si el diagrama de secuencia representa un escenario concreto (un caso concreto de ejecución) se denomina **diagrama de secuencias de instancias**.

Si el diagrama de secuencia representa un escenario genérico (todos los posibles casos de ejecución) se denomina **diagrama de secuencias genérico**. Puede construirse este diagrama a partir de los diversos escenarios, que pueden ayudar en su comprensión.

En este segundo caso, es necesario representar estructuras de contro (if-then-else o bucles). En la Ilustración 7 se representa un bucle WHILE. El mensaje lleva "*[condicion de mientras]" sobre la flecha y antes del nombre del mensaje. El mensaje es enviado varias veces mientras se cumpla la condición X.

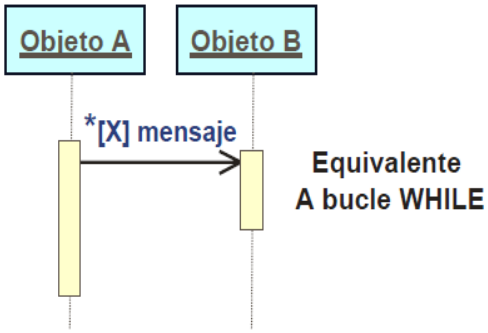

Ilustración 7: While

Si una condición produce una bifurcación, se representan las dos opciones indicando la condición entre corchetes como en la Ilustración 8.

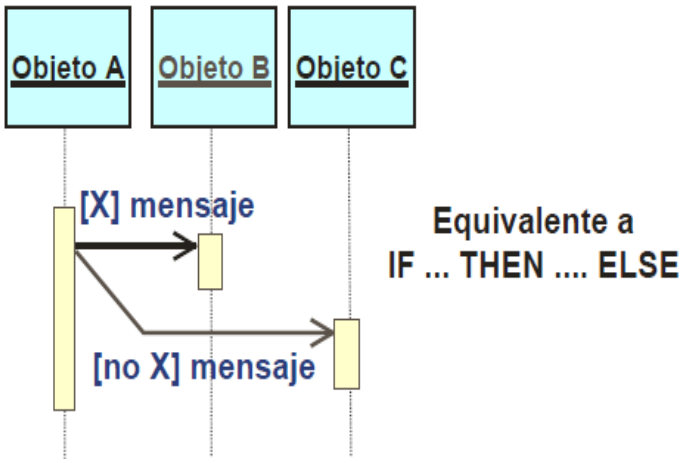

Ilustración 8: if-then-else

Cuando hay dos o más alternativas de ejecución en el lado del destinatario, se añade una bifurcación en la línea de vida del destinatario y se indica en el mensaje en qué caso o bajo qué condiciones se ejecuta cada una.

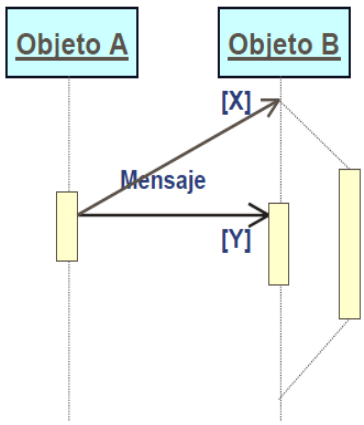

Ilustración 9: Alternativas del lado del destinatario

En la figura siguiente se muestra un ejemplo complejo de diagrama con varios de los elementos vistos hasta ahora, que representa una llamada hecha usando una Central de llamadas.

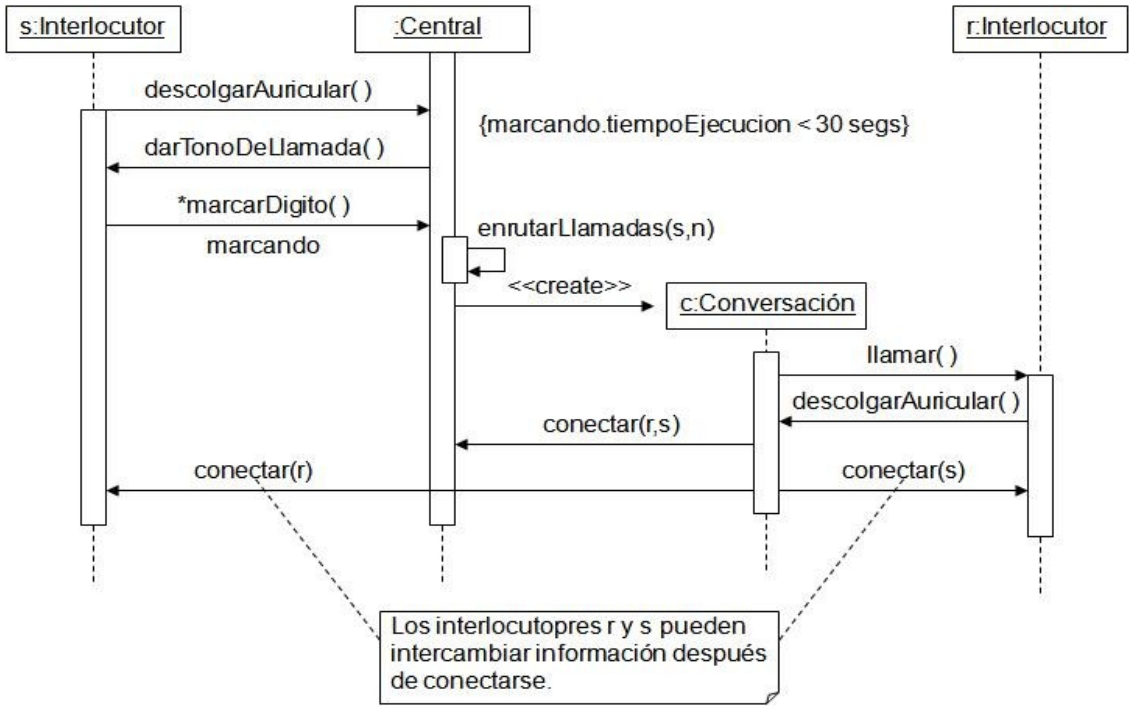

Ilustración 10: Llamada a través de una Central de Llamadas

Ejercicio 1
Con objeto de familiarizarte con la herramienta StarUML, dibuja el diagrama anterior con esa herramienta.

Por las peculiaridades de StarUML el diagrama debe quedar:

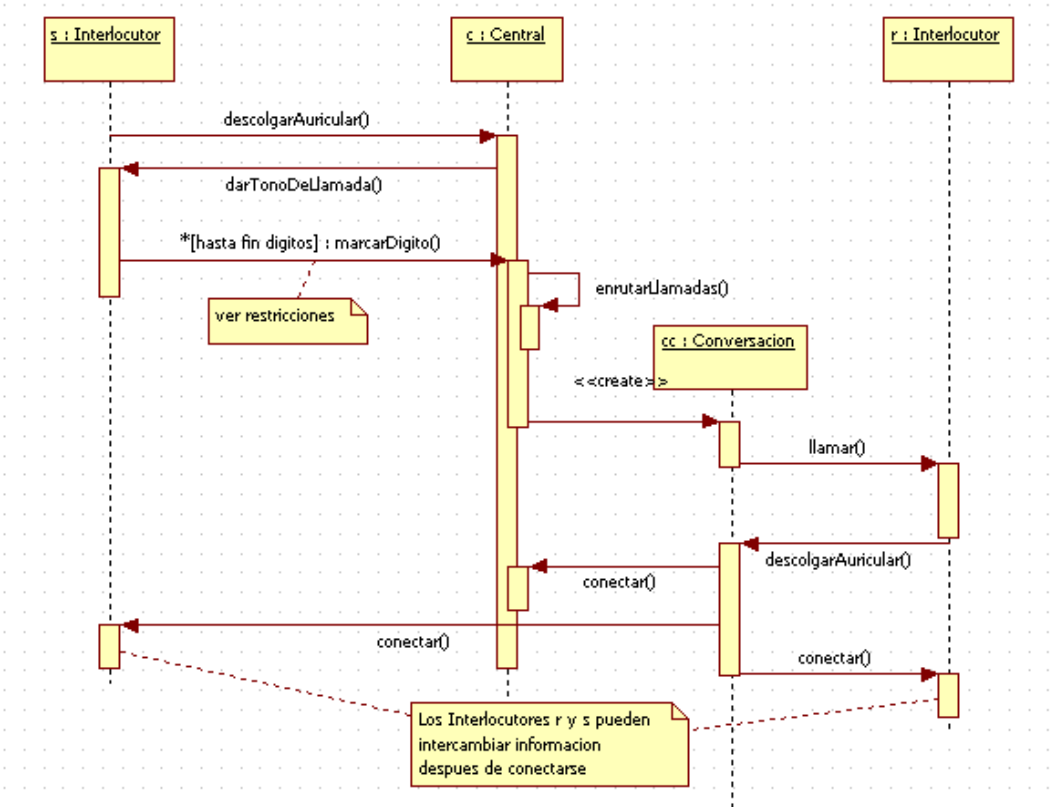

Ilustración 11: Diagrama anterior con StarUML

Ejercicio 2
Haz el diagrama de secuencia de instancias del termómetro del tema de diagramas de estado. Descarga el UML del diagrama de clases del termómetro de la web.

Ejercicio 3
Una máquina de refrescos se compone de tres partes: Frontal, Despachador y Registrador. Dibuja el escenario (diagrama de secuencias de instancias) que representa la interacción siguiente entre los objetos cuando una persona solicita un producto y sucede:

- *La persona introduce una moneda en el Frontal y pulsa el botón de un refresco*
- *El Frontal comunica el importe introducido y el producto seleccionado al Registrador.*
- *Como el importe es exacto y hay producto en existencias, el Registrador ordena al Despachador que libere un refresco del producto elegido.*

(Descarga el enunciado con el diagrama de clases y casos de uso desde la web)

Ejercicio 4
Al caso anterior, añade un escenario (diagrama de secuencias de instancias) con el camino de ejecución que se produce si el importe no es exacto y es superior al del producto seleccionado, sabiendo que el Registrador es el que calcula si hay que devolver o no cambio y el que ordena al Frontal la devolución del importe sobrante.

Ejercicio 5
Fusiona los ejercicios de los casos de los dos ejercicios anteriores en un único diagrama de secuencias genérico.

2 DIAGRAMAS de COLABORACIONES

Los diagramas de colaboraciones al igual que los de secuencias muestran la interacción entre objetos.

Sin embargo, no se organizan como los diagramas de secuencias, en el tiempo, sino en el espacio.

Para cada diagrama de secuencias puede realizarse un diagrama de colaboraciones equivalente y viceversa. En realidad, son dos maneras de representar el mismo concepto de interacción entre objetos.

En los diagramas de colaboraciones, al igual que en los de secuencias, se representan los objetos y los mensajes que se intercambian. Los mensajes se representan mediante una línea sobre la que se indica el mensaje enviado con sus parámetros si los tiene. Puede precederse el nombre del mensaje por la secuencia de ejecución del mismo separada por 2 puntos.

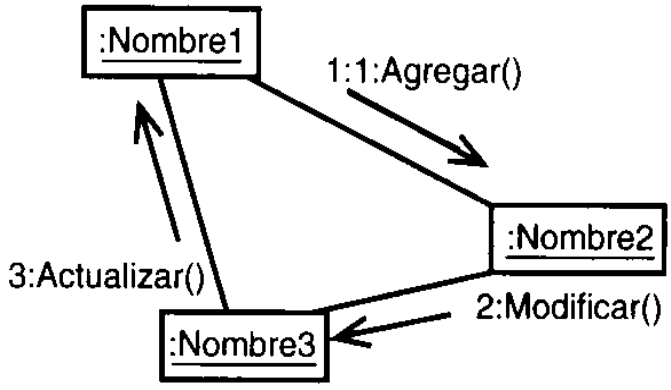

Ilustración 12: Ejemplo de diagrama de colaboraciones

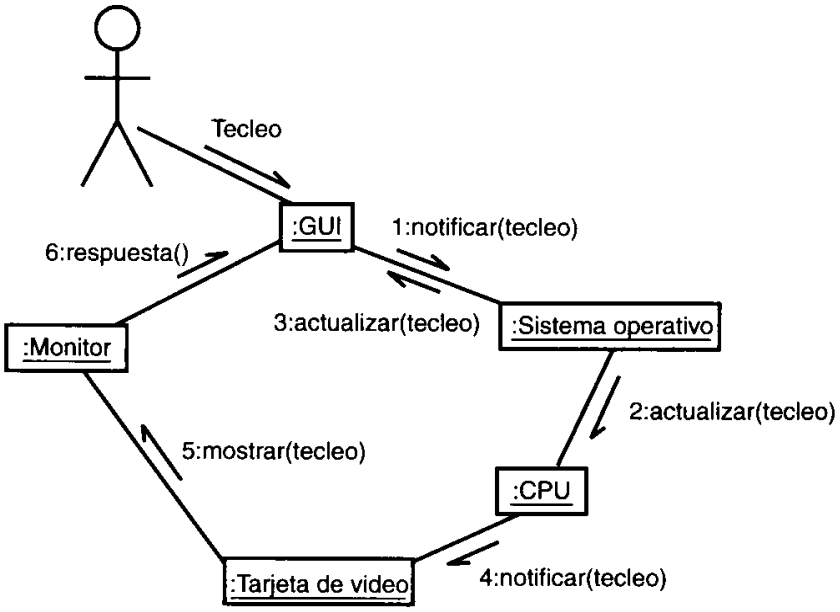

Ilustración 13: Ejemplo con mensajes asíncronos y actor

Es posible representar los distintos **estados** que atraviesa un objeto duplicando ese objeto e indicando entre corchetes el estado. Los rectángulos que representan los estados de un objeto se conectan mediante una flecha discontinua y el estereotipo <<se torna>>.

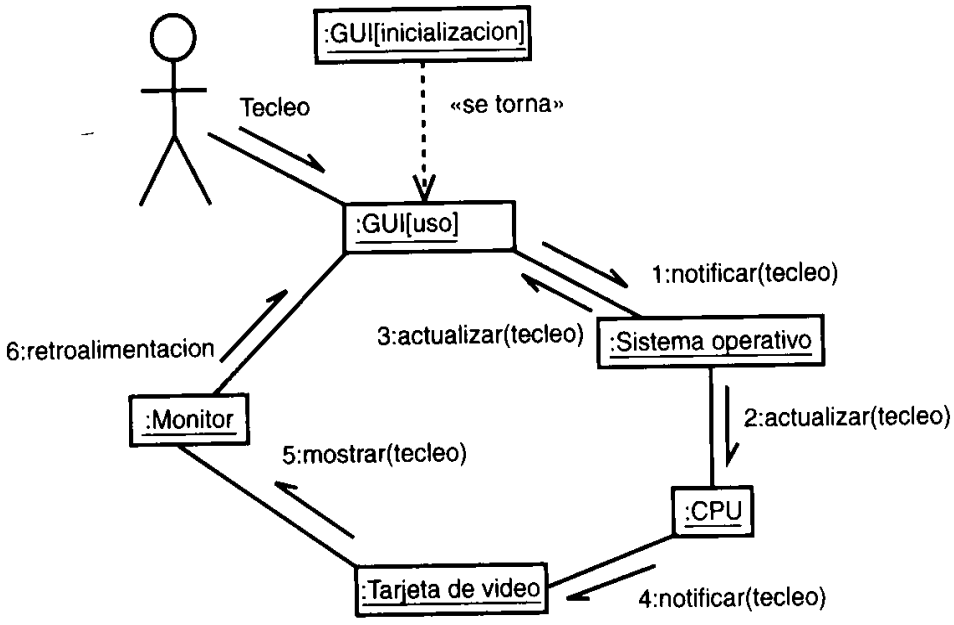

Ilustración 14: Estados del GUI

En una situación real será preciso considerar las alternativas derivadas de **evaluar una condición**. Esto es, ejecutar un mensaje u otro en función de si se cumple o no determinada condición.

En el diagrama de la Ilustración 12 se muestra el ejemplo de la máquina de refrescos donde hay tres

objetos **`Frontal`**, **`Registrador`** y **`Despachador`**. **`Frontal`** es el interfaz con el usuario, registra monedas y peticiones de productos. **`Registrador`** comprueba el importe introducido y las disponibilidades de productos y cambio. **`Despachador`** da al cliente el producto pedido y/o el cambio.

Las condiciones se representan entre corchetes, como en el diagrama de secuencia.

Los mensajes pueden numerarse de dos maneras. Como en el diagrama de secuencia, según orden de ejecución. Esta manera no refleja con claridad suficiente la posibilidad de ejecuciones concurrentes. Este tipo de numeración es el admitido por StarUML.

Hay otra posibilidad que es representar la numeración con anidaciones:

Cuando un mensaje es consecuencia de otro anterior, se numera como "n" al precedente y "n.1" al consecuente. El mensaje 1.1 es consecuencia del 1 y se ejecuta primero 1 y luego 1.1

Cuando hay condiciones excluyentes, éstas se reflejan entre corchetes, pero la numeración se pone acompañada por una letra. Los mensajes 1a y 1b serían excluyentes. Igual sucedería con los mensajes 3.4a y 3.4b

La numeración es consecutiva si los mensajes pueden ejecutarse de modo concurrente en el tiempo. 1 y 2 son mensajes concurrentes. 2.3 y 2.4 también.

En el ejemplo de los refrescos, el mensaje 1.1 es consecuencia del 1 y se ejecuta 1.1 después del mensaje 1. Los mensajes 1.1a y 1.1b son excluyentes y se ejecutan tras 1.1. El mensaje que sigue a 1.1a es 1.1a.1 y tras él se ejecuta 1.1a.1.1

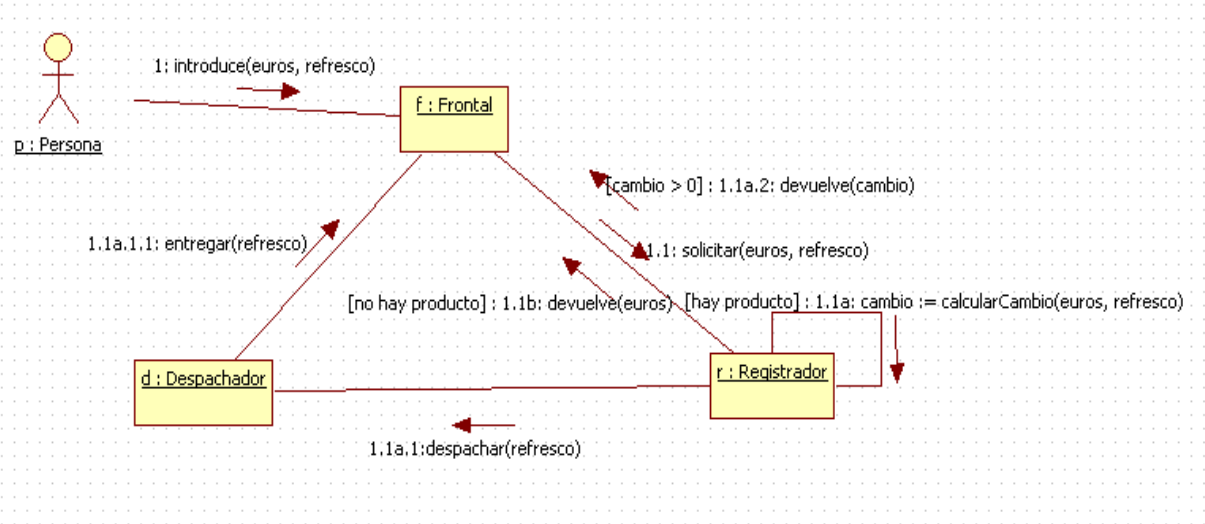

Ilustración 15: Máquina de refrescos con secuencias anidadas

La solución no anidada sería:

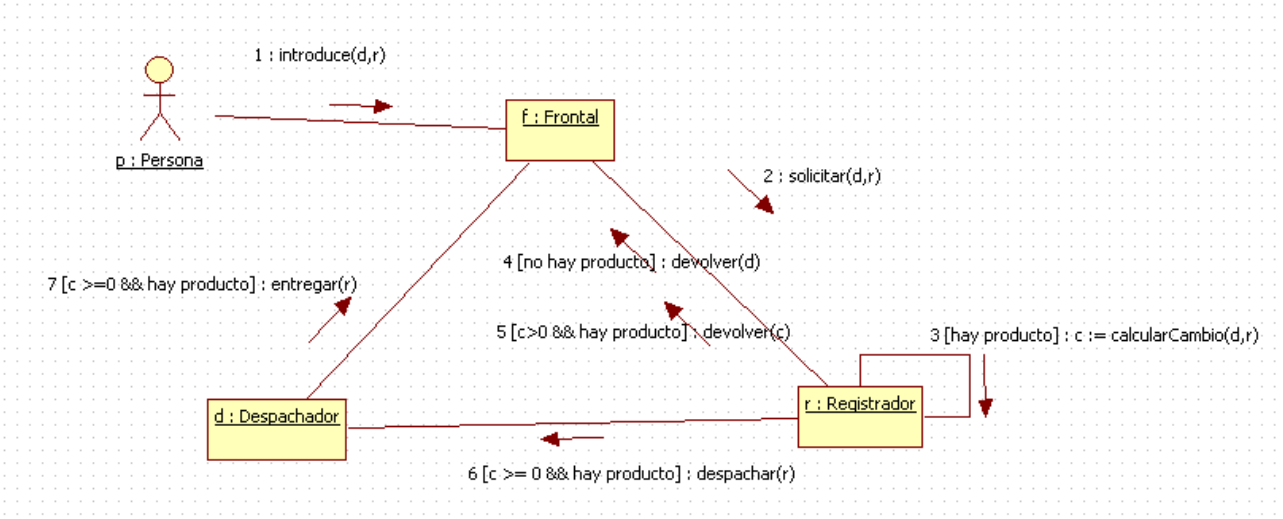

Ilustración 16: Máquina de refrescos sin anidación en la secuencia

En este último caso, hace falta indicar las condiciones en todos los mensajes afectados, cosa que no sucede en el diagrama anterior donde la numeración, al indicar consecuencia, permite no volver a indicar una misma condición para toda una "rama" de mensajes. En la Ilustración 16, el hecho de devolver cambio puede hacerse simultáneamente a la tarea de expender el producto, pero en este diagrama no queda evidente esa posibilidad.

Como en los diagramas de secuencia, pueden enviarse mensajes de creación y destrucción, que se representan precedidos por el estereotipo correspondiente <<crear>> o <<destruir>>.

Pueden así mismo representarse bucles While poniendo la condición entre corchetes precedidos éstos por un asterisco.

3 DIAGRAMAS de ACTIVIDADES

Son una extensión del diagrama de estados visto anteriormente. Al igual que los diagramas de estados, comienzan con una bola negra y finalizan con una "diana", pero al contrario que aquellos[1], aquí se representan las acciones dentro de un rectángulo de esquinas redondeadas.

Son equivalentes a los antiguos diagramas de flujo del análisis y diseño estructurado.

Aquí solo veremos la simbología más común. Puede representarse:

- Secuencia
- Decisión
- Concurrencia

3.1 Secuencia

Se representan dos acciones que suceden una antes de la otra en un orden temporal.

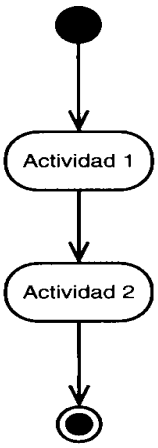

Ilustración 17: Secuencia

3.2 Decisión

Representa caminos alternativos que se toman en función del cumplimiento de una condición.

[1] Ls acciones en diagramas de estados podían ser transiciones o acciones realizadas al entrar, salir o durante un estado.

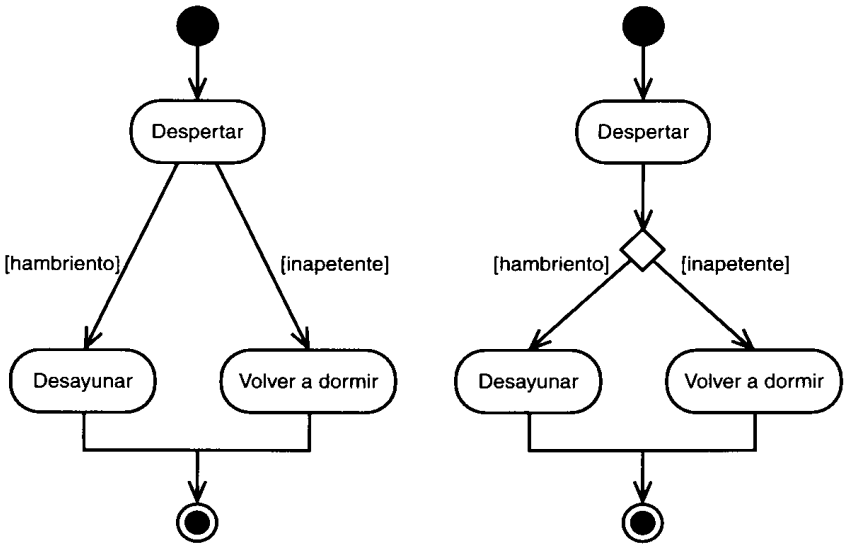

Ilustración 18: Decisión

3.3 Concurrencia

Actividades que se producen o ejecutan simultáneamente en el tiempo. Se representa con una barra horizontal o vertical.

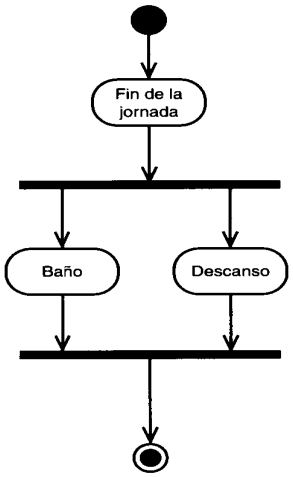

Ilustración 19: Concurrencia

Los caminos concurrentes pueden sincronizarse al final, o simplemente terminar con un símbolo de final de flujo (un círculo con una X). El final de flujo representa el final de la ejecucuión de ese camino, pero no el final de la ejecución del proceso completo.

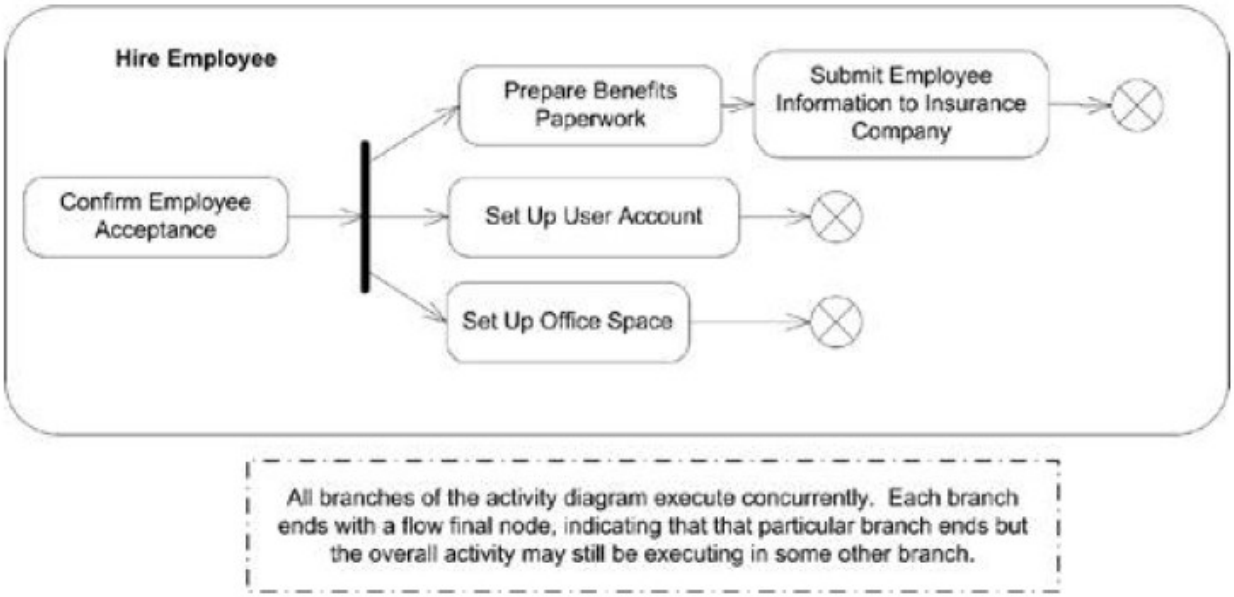

Ilustración 20: Caminos con final de flujo

Un diagrama de actividades de ejemplo:

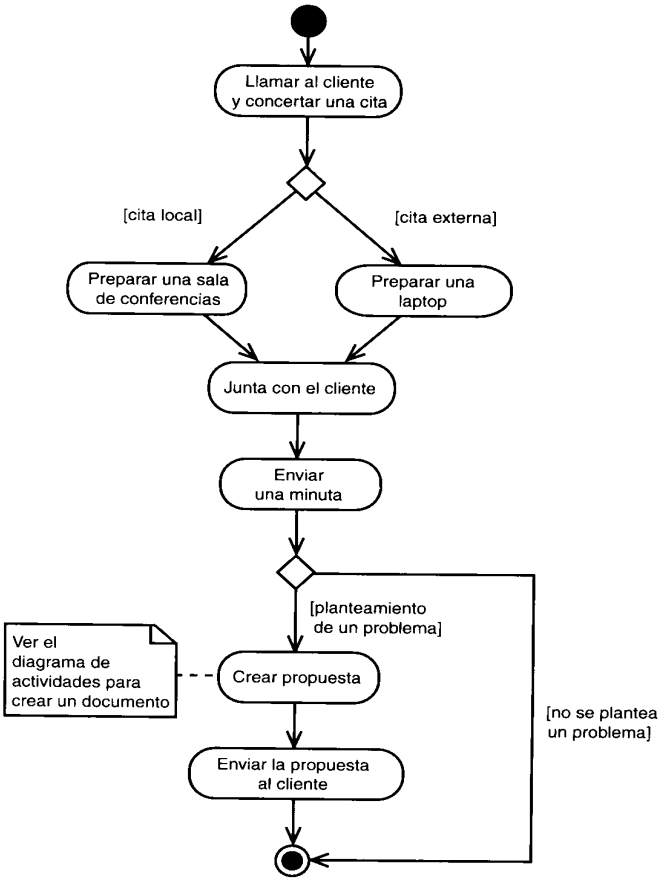

Ilustración 21: diagrama de actividades "Cita"

Ejercicio 6
Elabora el diagrama de actividades correspondiente al caso de la máquina de refrescos.

4 DIAGRAMAS de COMPONENTES

Un **componente de software** es una parte física de un sistema, por ejemplo una tabla relacional, un archivo de datos en el disco, un archivo ejecutable, una biblioteca de vínculos dinámicos, documentos, etc.

Las clases pueden almacenarse físicamente en diferentes archivos, puede ser interesante indicar en qué lugares físicos están. De ese modo los componentes pueden también ser reutilizados en otros proyectos.

Los componentes, pese a ser una parte física, tienen en común con la parte "lógica" del proyecto que poseen interfaces, comprendidos éstos en el mismo sentido que en las clases. Los interfaces son una "firma" que es pública y conocida por otros componentes (o por otras clases en el diseño lógico) y que pueden ser utilizados por ellos.

La relación entre componente e interfaz se denomina **realización**, como en el caso de las clases.

El componente que provee los servicios utilizables a través de un interfaz se dice que provee una **interfaz de exportación**. El componente que utiliza los servicios de otro componente, se dice que utiliza un **interfaz de importación**.

Un componente puede ser rediseñado y/o sustituir a uno antiguo si posee el mismo aspecto externo, la misma firma, esto es, el mismo interfaz. Esto evita tener que ver el interior del componente y su código para hacer una operación de sustitución o mejora de partes del proyecto. Si la firma (interfaces) es la misma, no será necesario, ahorrándose un coste considerable en horas de trabajo.

4.1 Tipos de componentes

1. De **distribución**. Son los archivos ejecutables: DLL, EXE, controles Active X o Java Beans.
2. **Componentes para trabajar en el producto**. Son aquellos a partir de los cuáles se crean los componentes del punto 1. Archivos de código o archivos de bases de datos, por ejemplo.
3. **Componentes de ejecución**, se crean como resultado de una ejecución. Por ejemplo archivos de datos.

4.2 Representación de un componente

Se representa como un rectángulo que tiene dos pequeños en el lado izquierdo. Dentro del rectángulo principal se coloca el nombre del componente.

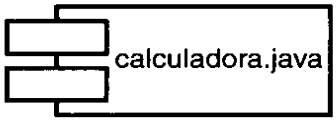

Ilustración 22: componente

Los componentes se agrupan en **paquetes**. Puede precederse el nombre de un componente con el nombre del paquete en el que se incluye.

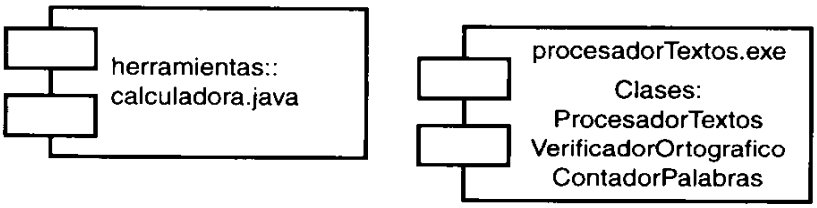

Ilustración 23: componentes precedidos por el nombre de paquete. A la derecha, información interna del componente.

En el componente de la derecha "procesadorTextos.exe" se añade a la derecha información que muestra los detalles del componente.

Otra forma de representar esto último es la siguiente:

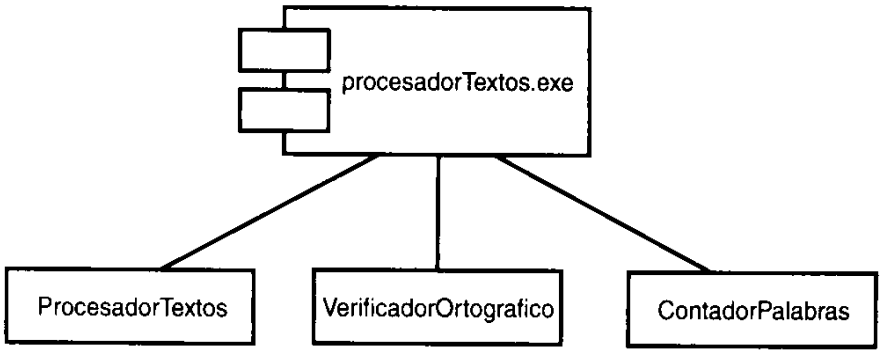

Ilustración 24: contenido de un componente

4.3 Representación de Interfaces

Hay dos maneras:

1) Representando la interfaz al estilo indicado en los diagramas de clases, con una línea discontínua y punta vacía hacia el interfaz desde el componente.

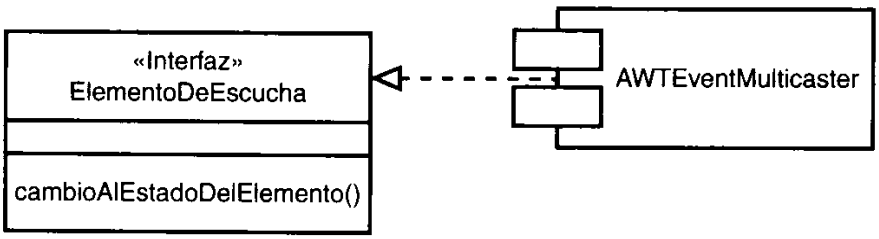

Ilustración 25: Interfaz de un componente

2) La forma alternativa es mediante un círculo que representa al interfaz conectado con el componente:

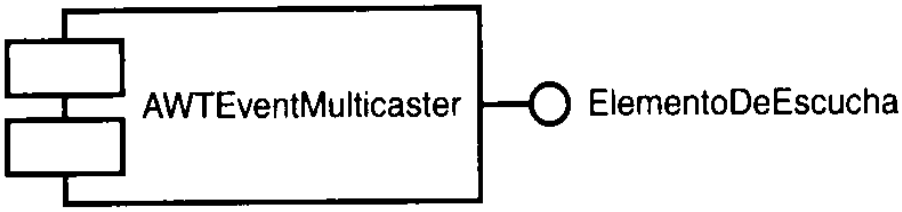

Ilustración 26: Representación del interfaz mediante bola

También puede representarse la dependencia (uso de un interfaz) como en el caso de las clases:

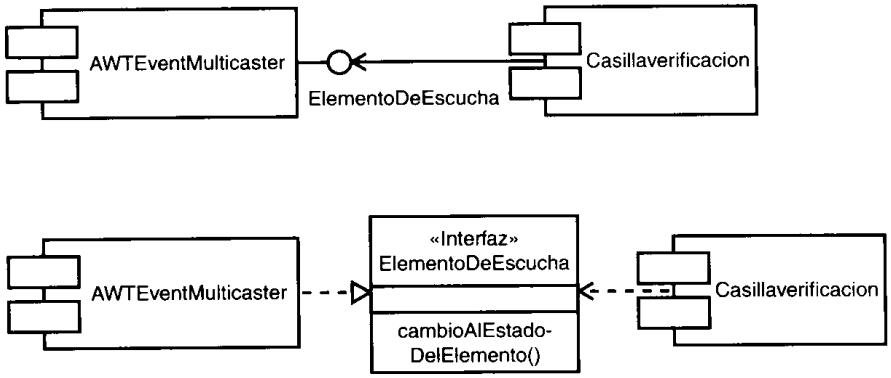

Ilustración 27: representaciones alternativas de interfaz, realización y dependencia

Si un componente necesita algo que está en otro componente, se unen con una flecha de dependencia desde el componente que requiere hacia el componente requerido.

En este caso. Si tenemos que los archivos .class son el código objeto cuyo fuente está en el correspondiente archivo .java, y tenemos que los componentes "Craps" usan la clase "Die", y que Craps.java usa el interfaz "ActioListener" de un paquete importado llamado java.awt.event, el gráfico es el siguiente:

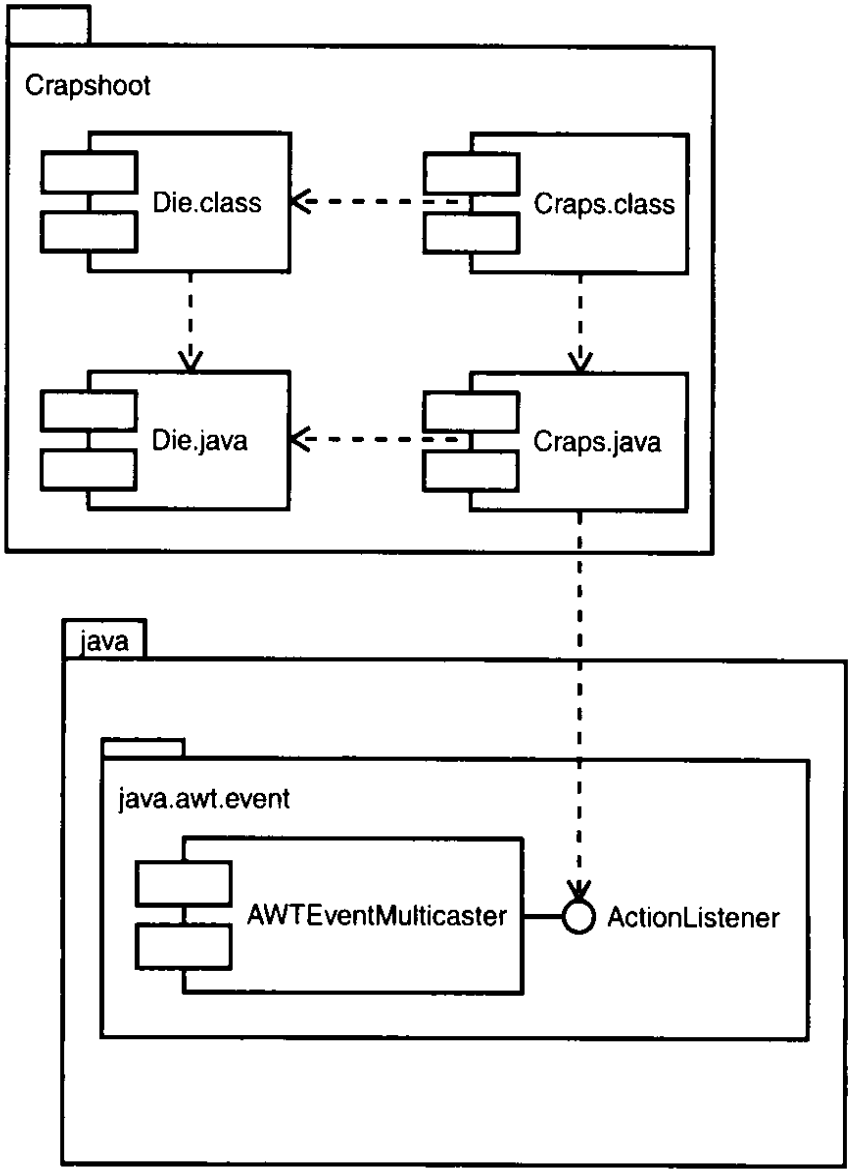

Ilustración 28: Ejemplo de Diagrama de componentes

Como vemos los paquetes se representan como clasificadores. La relación de dependencia entre paquetes puede representarse de la siguiente manera:

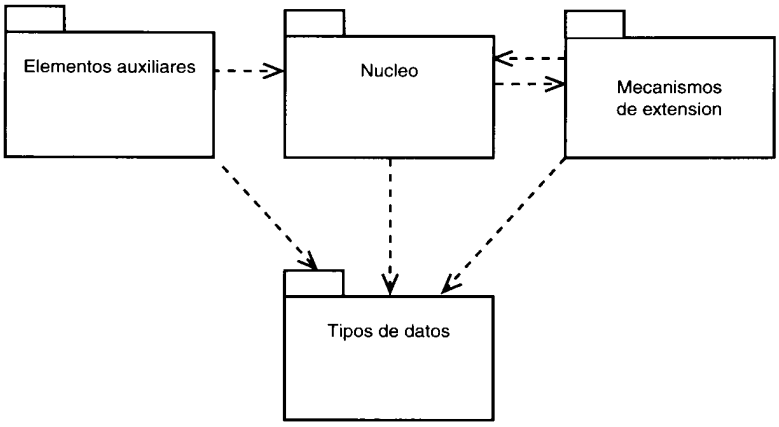

Ilustración 29: Dependencia entre paquetes.

Ejercicio 7
En un proyecto se tienen dos paquetes A y B. A contiene los componentes "a", "b", "c" y B contiene los coponentes "d" y "e". Dibuja el diagrama de componentes si se tiene que:

- *El componente "a" necesita o requiere al componente "d" y al "b"*
- *"c" provee un interfaz de exportación "Y" que es usado por "d"*
- *"d" requiere a "e"*

5 BIBLIOGRAFÍA Y ENLACES

- Aprendiendo UML en 24 horas, Joseph Schmuller , Prentice-Hall
- UML 2.0 In a Nutshell, Junio 2005, Dan Pilone, Neil Pitmande, O'Reilly
- www.omg.org (Object Management Group).
- http://www.omg.org/technology/readingroom/UML.htm
- http://msdn.microsoft.com/es-es/library/dd409377.aspx