Contenido

Página

1 UML en el desarrollo

Al iniciar un proyecto de software es esencial captar los requerimientos que debe cumplir el sistema. Un elemento clave para lograr esto es establecer una comunicación apropiada con el usuario, a fin de comprender plenamente sus necesidades. Para ello UML provee los **casos de uso**.

Debido a que los casos de uso ayudan a captar los requerimientos del sistema, establecen el marco estructural (fig. 2) sobre el cual se apoyarán las diferentes vistas del sistema, tanto en el modelo lógico como en el físico.

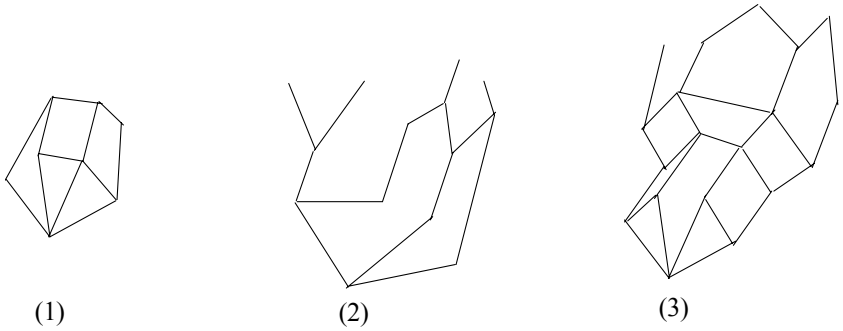


fig. 2 Marcos que establecen las bases para la estructura de los sistemas

Un caso de uso es una descripción de la funcionalidad ofrecida por el sistema bajo un uso específico de interacción con un actor externo (usuario u otro sistema). La suma de todos los casos de uso dan la visión externa del sistema y representan su comportamiento global. En la fig. 3 se representa el marco del elemento (2) de la fig. 2 Este marco ha sido construido a partir del comportamiento esperado del sistema, pero ahora se han etiquetado los comportamientos y subcomportamientos internos.

En la figura se indica que **a, b, c, d** son comportamientos de interés para el usuario y por medio de los cuales interactúa con el sistema, estos se identifican como casos de uso y se representan en el *diagrama de casos de uso* que aparece en el lado derecho de la figura. Esto corresponderá a una vista de la arquitectura externa del sistema.

Para poder llevar a cabo esos comportamientos, es necesario definir subcomportamientos internos. Podemos observar que para obtener el comportamiento **a** se deberán ejecutar **e, h** y **e, i**; de forma semejante sucederá con los comportamientos **b , c** y **d**.

Cada comportamiento y subcomportamiento involucra un conjunto de operaciones necesarias para obtener la funcionalidad requerida; por tanto un siguiente paso implicaría identificar tipos de objetos por necesidad de la aplicación[1], encargados de llevar a cabo esas operaciones y, como consecuencia, responsables de tales comportamientos.

[1] Unos que forman parte del dominio del problema y otros que son creados para encaminarse hacia la solución.

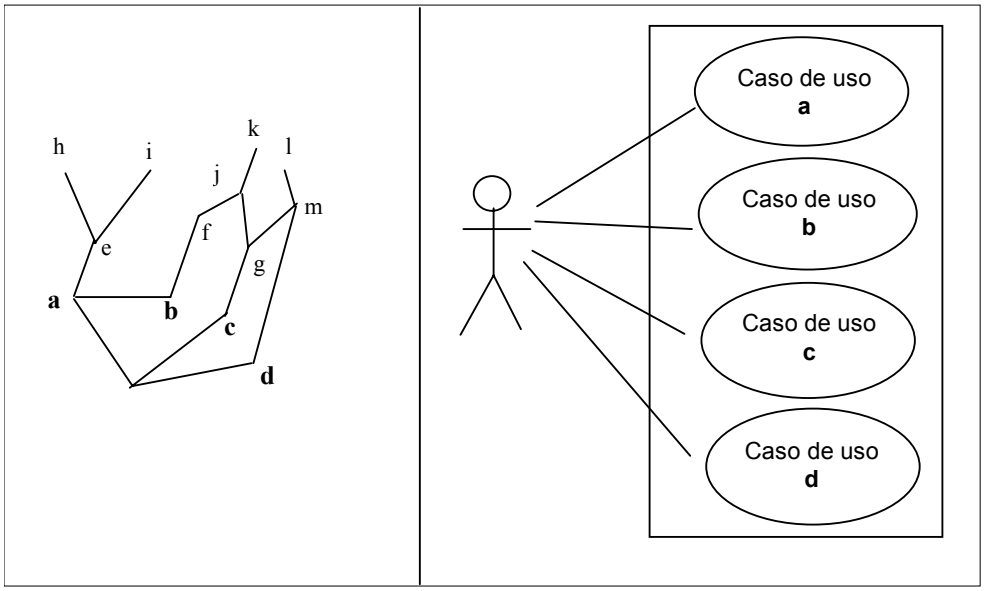


fig. 3 Comportamientos de un sistema y su representación en un diagrama de casos de uso

En la fig. 4 se representan los objetos que han sido asignados con los diferentes comportamientos que el sistema debe tener. Se han etiquetado con el nombre del objeto y el tipo de objeto al que pertenece. Si analizamos brevemente los objetos involucrados en la obtención del comportamiento **a**, observamos que es la colaboración de los objetos (a´, A), (e´, E), (h´, H), (i´, H) lo que permite su obtención, siendo (a´, A) el objeto a través del cual se lleva a cabo la solicitud.

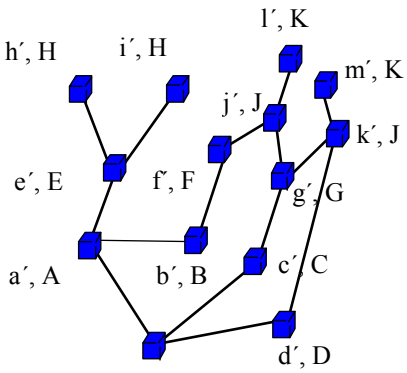


fig. 4 Objetos a los que se les han asignado responsabilidades

Un aspecto que vale la pena mencionar en este ejemplo, es que los comportamientos **h** e **i** son comportamientos semejantes que se asignan a dos objetos de un mismo tipo, esta puede ser una condición alrededor del análisis del dominio del problema o posterior, debido a aspectos de diseño.

Podemos auxiliarnos de los *diagramas de actividad* (fig. 5) para mostrar el flujo de actividades que se desarrollan en la ejecución de un caso de uso y el resultado que se espera. Un aspecto clave de este tipo de diagrama es que ofrece la posibilidad de representar procesamiento paralelo, con lo que se permite a los desarrolladores no limitarse al procesamiento secuencial en aquellas

situaciones donde existe la oportunidad de hacer cosas en paralelo, con ello se mejora la eficiencia del procesamiento o se revela que no hay un orden pre-establecido para algunas actividades.

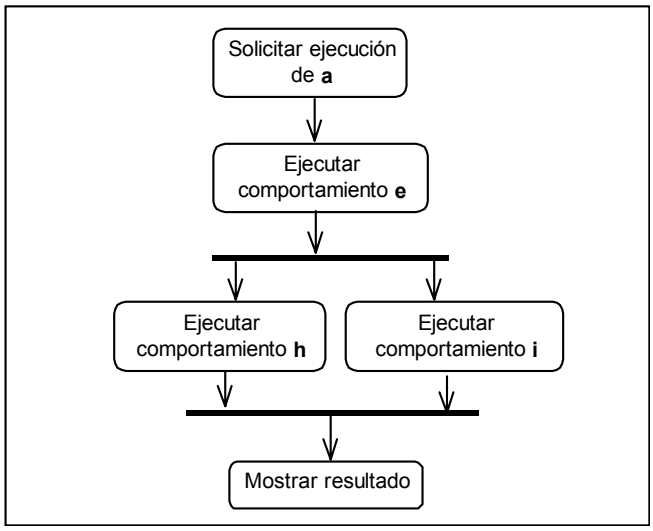


fig. 5 Diagrama de actividad

Utilizando un *diagrama de clases* podemos reflejar los tipos de objetos involucrados (para abreviar, solamente los tipos de objetos involucrados en el comportamiento **a**).

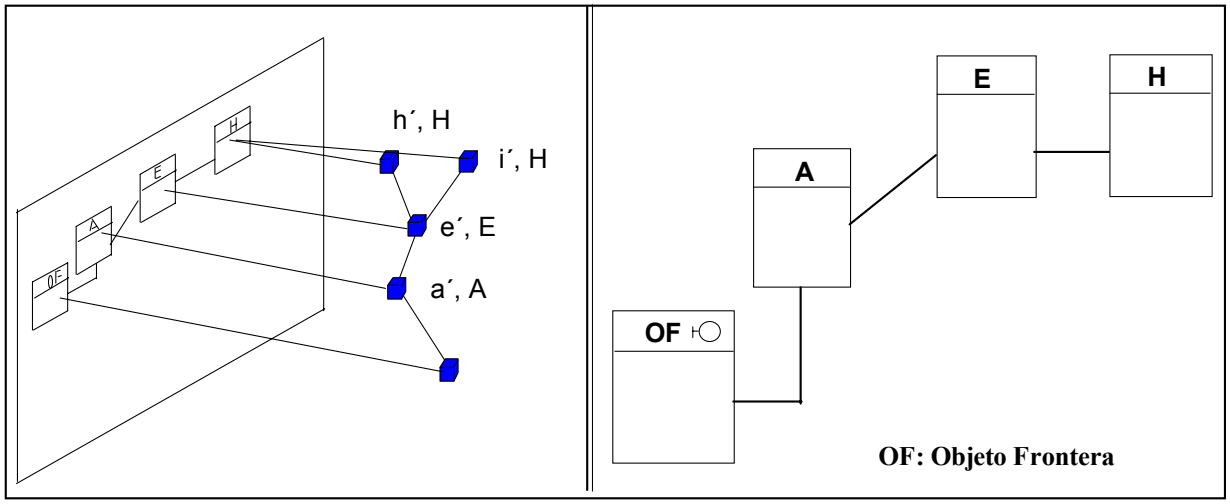


fig. 6 Diagrama de clases

En el diagrama de clases se muestran los tipos de objetos que se asocian y los tipos de relaciones establecidas, que a su vez representan vías de comunicación a través de las cuales circularán los mensajes para lograr los comportamientos esperados. Esto nos presenta una vista estática del sistema.

Utilizamos ya sea un *diagrama de secuencia* o *diagrama de colaboración* para observar y analizar las interacciones que se producen entre los diferentes componentes. La fig.7 nos muestra

una representación típica del modelo, lo cual correspondería a una vista dinámica parcial del sistema.

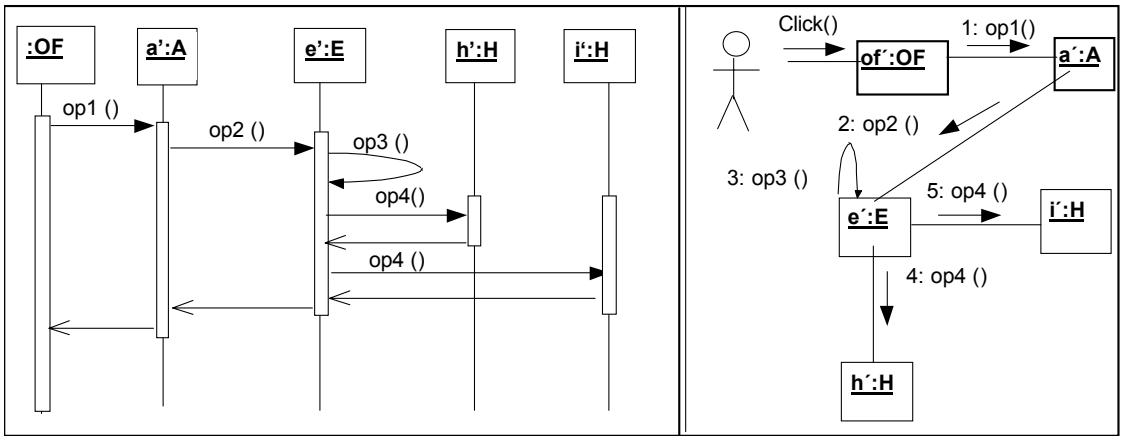


fig. 7 Diagrama de secuencia y diagrama de colaboración con la misma interacción

Podemos emplear el *diagrama de paquetes* para tener una vista de la estructura organizativa del sistema observando cómo se encuentran agrupadas las clases relacionadas y las dependencias que existen entre estos grupos denominados paquetes.

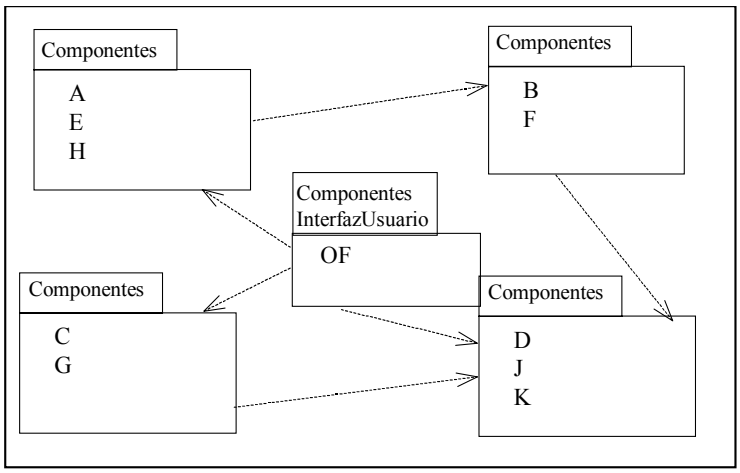


fig. 8 Diagrama de paquetes

Los diagramas mencionados se utilizan a través de las diferentes fases de análisis y diseño, donde los modelos se van refinando hasta que sean susceptibles de ser implementados, momento en el cual podemos emplear *diagramas de despliegue* y *diagramas de componentes* para mostrar aspectos de implementación que resulten trascendentes de ser modelados.

2 Elementos para modelar la arquitectura

Modelar la arquitectura implica establecer el esqueleto que soporta el modelo esencial o conceptual del sistema y agregar la infraestructura necesaria a través de los modelos subsecuentes, que harán que una aplicación funcione.

Puede distinguirse entre arquitectura externa y arquitectura interna. UML ofrece los diagramas de casos de uso para especificar la vista externa del sistema en términos de servicios y los diagramas de paquetes para agrupar elementos de un modelo.

2.1 Diagrama de Casos de Uso

El caso de uso es una técnica de modelaje que se emplea con el fin de describir las funciones de un sistema. Un caso de uso es una descripción del conjunto de secuencias de acciones, incluyendo variantes, que un sistema ejecuta para entregar un resultado observable y de cierto valor para un actor [Booch 1999].

Como cada caso de uso describe un cierto comportamiento de interés para el usuario, el conjunto de todos los casos de uso encontrados deben describir toda la funcionalidad que se espera del sistema.

El diagrama de casos de uso es el medio que UML provee para representar los casos de uso, es decir, los diferentes comportamientos que el usuario espera le proporcione el sistema. Un diagrama de casos de uso está compuesto de 4 elementos: actor, asociación, caso de uso y sistema (fig. 9).

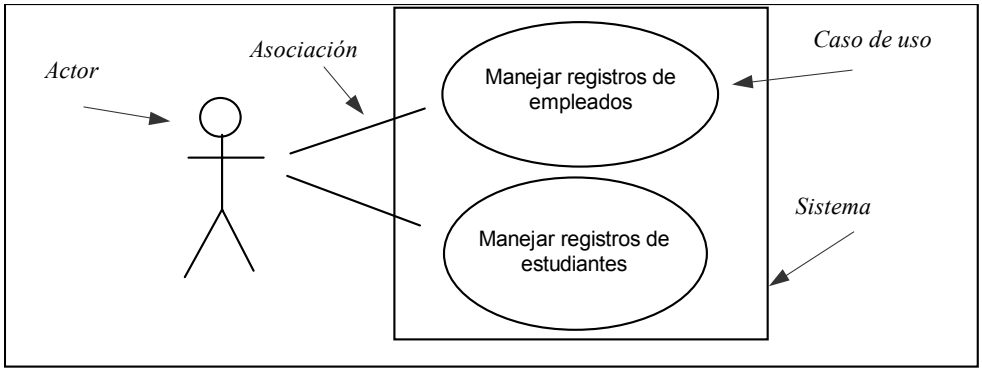


fig. 9 Diagrama de casos de uso

El flujo de eventos que genera un caso de uso normalmente se describe textualmente y consiste en una especificación de cómo el actor y el caso de uso interactúan. Normalmente la descripción de un caso de uso incluye:

- Objetivos del caso de uso
- Cómo es iniciado el caso de uso
- El flujo de mensajes entre el actor y el sistema
- Flujos alternativos en el caso de uso

- Cómo finaliza el caso de uso con el valor de interés para el actor.

Un caso de uso puede también ser descrito a través de un diagrama de actividad (fig. 10), el cual presenta la secuencia de actividades, su orden y, opcionalmente, decisiones que se lleven a cabo e indiquen cuál es la siguiente actividad por ejecutarse.

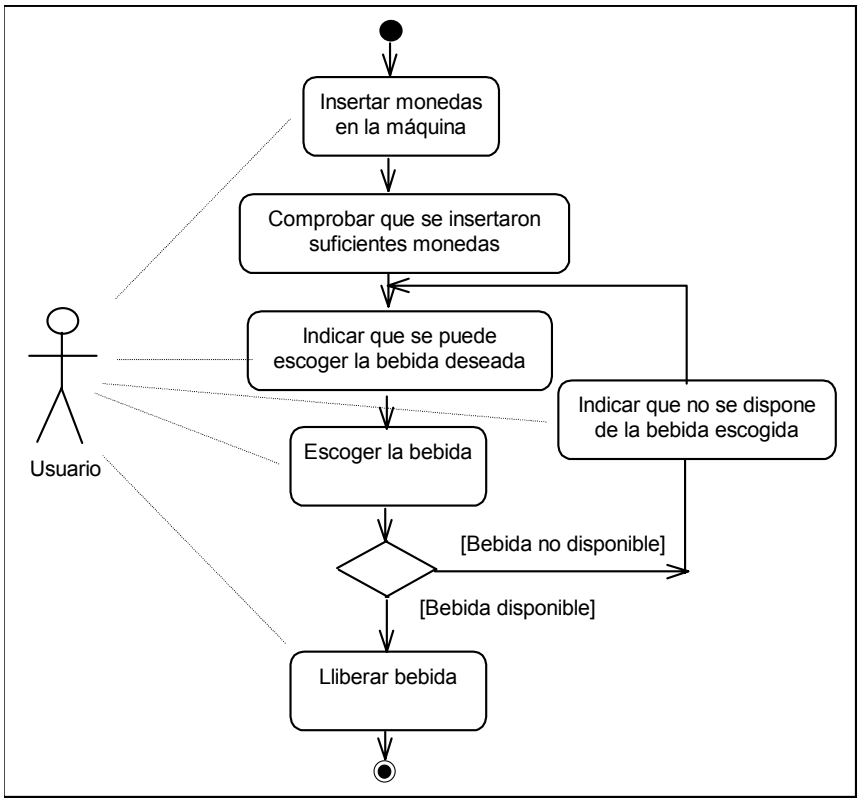


fig. 10 Diagrama de actividad para describir un caso de uso

Concepto	Descripción	Fig.
Actor	Cualquier entidad externa que tiene algún interés de interacción con el sistema.	9 11
Caso de uso	Funcionalidad llevada a cabo por el sistema desde que es iniciada por el actor hasta que se entrega un resultado de interés para ese actor.	9
Sistema	En el contexto de casos de uso, es visto como una "caja negra" que provee los diferentes casos de uso.	9
Asociación	Relación que muestra cuál actor interactúa con cuál caso de uso.	9
Extiende	Relación entre casos de uso donde un caso de uso extiende a otro, al agregar acciones a un caso de uso más general.	12 13
Incluye	Relación entre casos de uso que implica que uno de los casos de uso incorpora explícitamente el comportamiento del otro caso de uso. El caso de uso incluido nunca es completo (autosuficiente), siempre forma parte de un caso de uso mayor que le sirve de base.	12 13

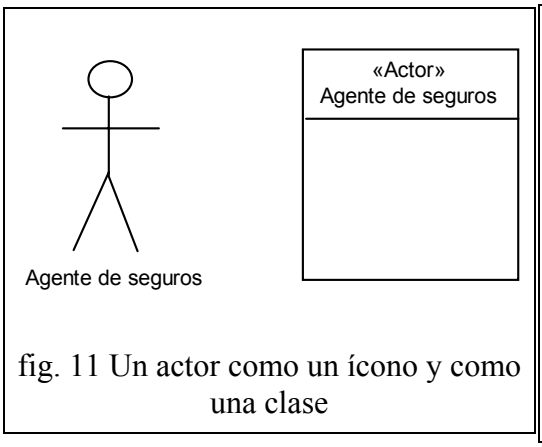


fig. 11 Un actor como un ícono y como una clase

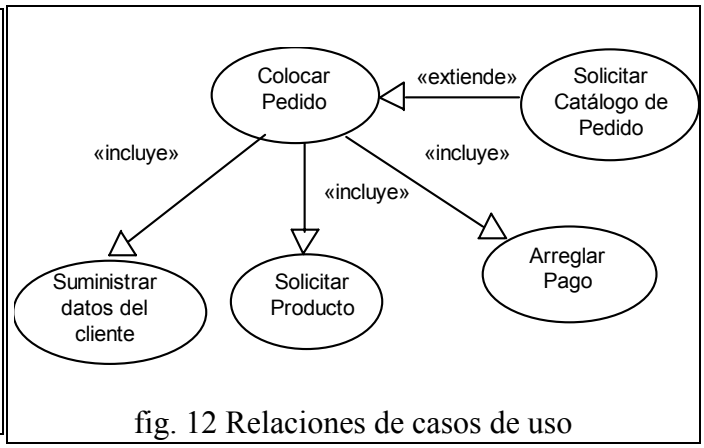


fig. 12 Relaciones de casos de uso

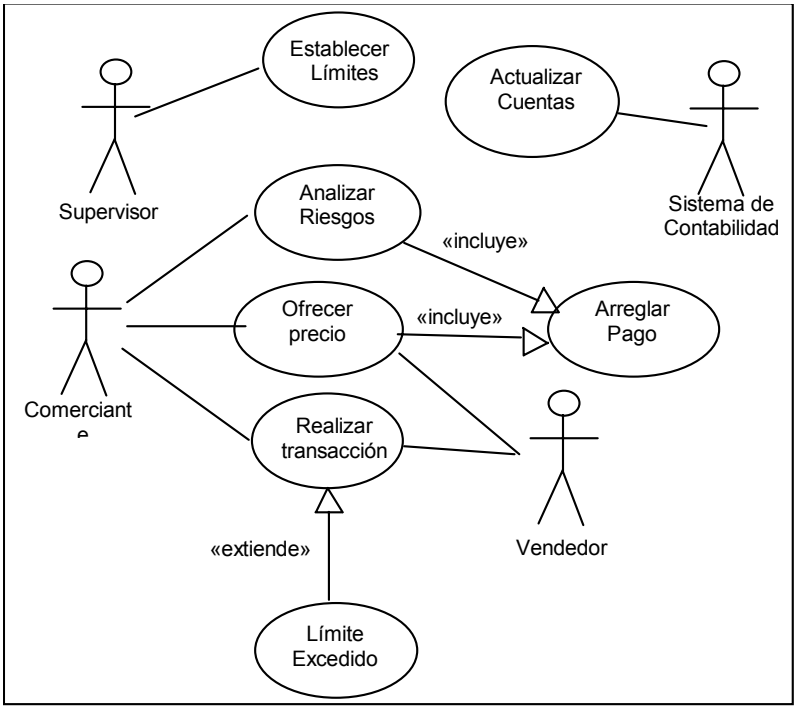


fig.13 Casos de uso y sus relaciones

2.2 Diagrama de Paquetes

Los paquetes permiten organizar un sistema grande en grupos de elementos de alguna forma relacionados. Los diagramas de paquetes muestran los paquetes y las dependencias que entre ellos existen. Normalmente los elementos que componen los paquetes consisten de las clases que soportan la aplicación, sin embargo su uso no se limita a ese tipo de elementos.

Estos diagramas son una herramienta importante para mantener control sobre la estructura global del sistema.

Cuando se presenta un paquete, su nombre se coloca dentro del rectángulo principal. Sin embargo, cuando se listan las clases que componen el paquete dentro del rectángulo principal el nombre del paquete se coloca en la pestaña. (fig. 15)

Concepto	Descripción	Fig.
Paquete	Mecanismo de agrupación para organizar elementos en grupos semánticamente relacionados.	14
Dependencia	Relación que existe entre dos elementos si cambios en la definición de uno de los elementos puede causar cambios en el otro elemento.	14
Generalización	Relación donde se establece que un paquete específico debe ajustarse a la interfaz de un paquete general.	15
Visibilidad	Nivel de acceso al contenido de un paquete. **Privado**: Solamente el paquete propietario. **Protegido:** Solamente el paquete propietario o sus descendientes pueden accederlo (por generalización-especialización). **Público:** Cualquier otro paquete puede acceder el contenido del paquete, vía dependencias «import» o «access». **Implementación:** El contenido de un paquete no puede ser importado. La diferencia entre la visibilidad protegida y privada consiste en que solamente los paquetes especializados (descendientes) pueden utilizar los elementos protegidos de un paquete de generalización	

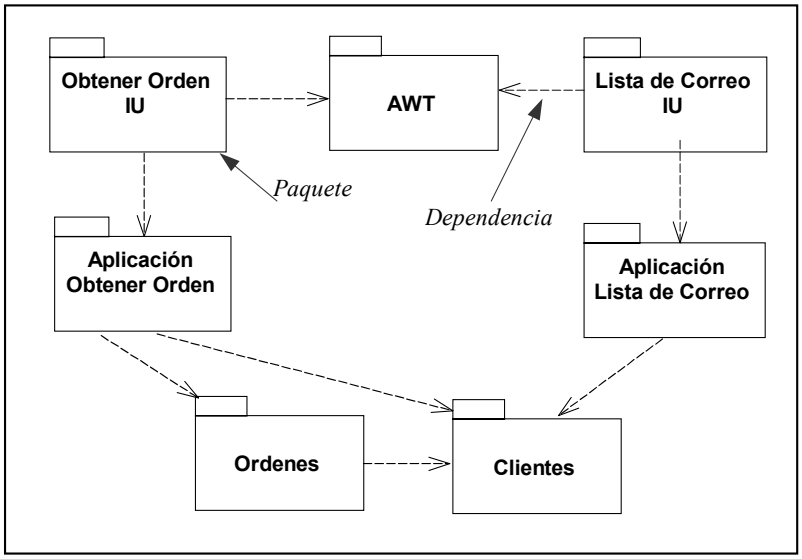


fig. 14 Diagrama de paquetes

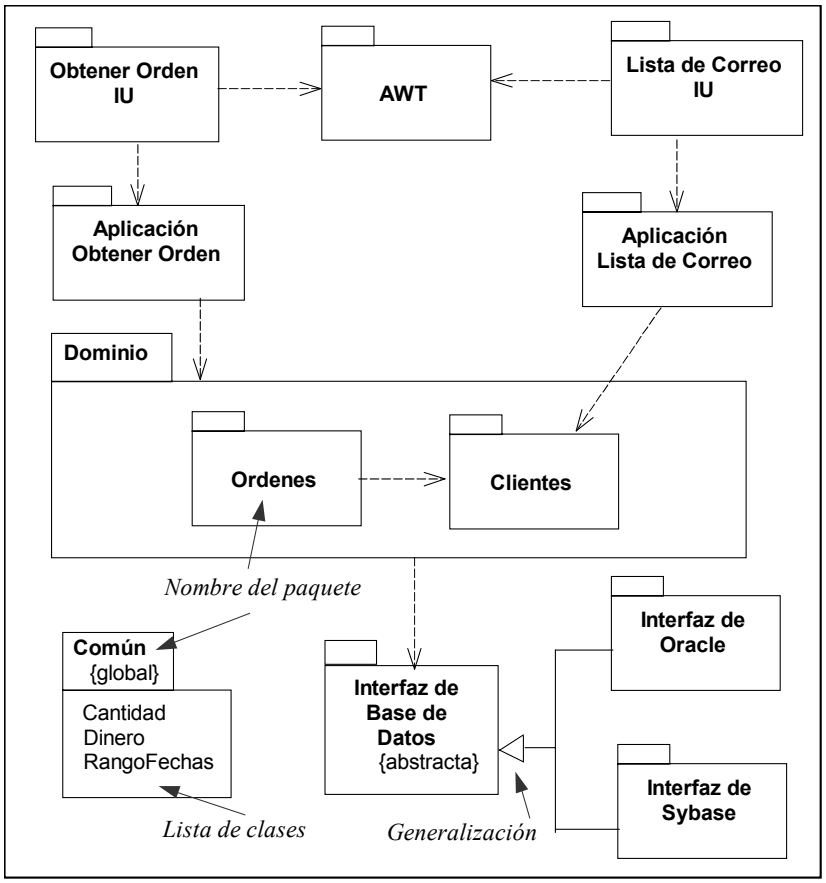


fig. 15 Diagrama de paquetes avanzado

3 Elementos para modelar la vista estructural (estática)

UML provee diagramas de clases y diagramas de objetos, los cuales se componen de elementos que permiten crear la vista estructural del sistema que se desear construir.

3.1 Diagrama de Clases

Un diagrama de clases nos presenta tanto los tipos de objetos que tendrán a su cargo la responsabilidad del comportamiento que el sistema ofrece, como los componentes internos (atributos y operaciones) de cada uno de ellos. Estos diagramas muestran también las múltiples relaciones que se establecen entre dichas entidades y que dan como resultado las estructuras colaborativas.

Al interpretar un diagrama de clases, es importante observar qué fase del modelaje representa. Como se mencionó en [Trejos 1999], al inicio del proceso tendremos un *modelo esencial*, donde los tipos de objetos representarán sus contrapartes en el dominio del problema. Posteriormente en el *modelo de especificación*, representamos tipos de objetos que incluyen aspectos orientados hacia la solución, se definen interfaces que persiguen obtener el comportamiento que el software debe proveer y por último en el *modelo de implementación* tenemos una representación con mayor nivel de detalle que incluye aspectos de implementación.

En la fig.16 se muestra un diagrama de clases. En él aparece un conjunto de clases que se relacionan de diferentes formas (asociación y generalización). Se incluyen también otros conceptos que apoyan el modelaje de la vista de estructura.

Los conceptos básicos que se utilizan para crear un diagrama de clases comprenden: clase, atributo, operación, asociación, multiplicidad, generalización/especialización, agregación, composición y rol.

Concepto	Descripción	Fig.
Clase	Una descripción de un tipo de objeto.	16
Atributo	Propiedad de un objeto.	16
Operación	Servicio proveído por un objeto.	16
Asociación	Conexión semántica entre instancias de clases.	16
Multiplicidad	Concepto que indica: para un objeto de un tipo, cuántos objetos del otro tipo participan en la relación. (1 Obligatoria), (0..1 Opcional), (* Multivaluada)	16
Navegabilidad	Condición que restringe la asociación en una dirección.	16
Generalización/ Especialización	Relación entre un tipo general y otro específico.	16
Agregación	Relación en la que un objeto está compuesto por otros.	17
Composición	Relación de agregación más estrecha, en la que las partes se encuentran dentro del todo.	17
Rol	Papel que desempeña una clase en una asociación.	16

Al presentar mayor complejidad lo que se desea modelar, un diagrama de clases puede incluir conceptos más avanzados: refinamiento, calificador, clase paramétrica (plantillas de clases genéricas), instanciación de clase paramétrica, clase abstracta, asociación o atributo derivado, visibilidad.

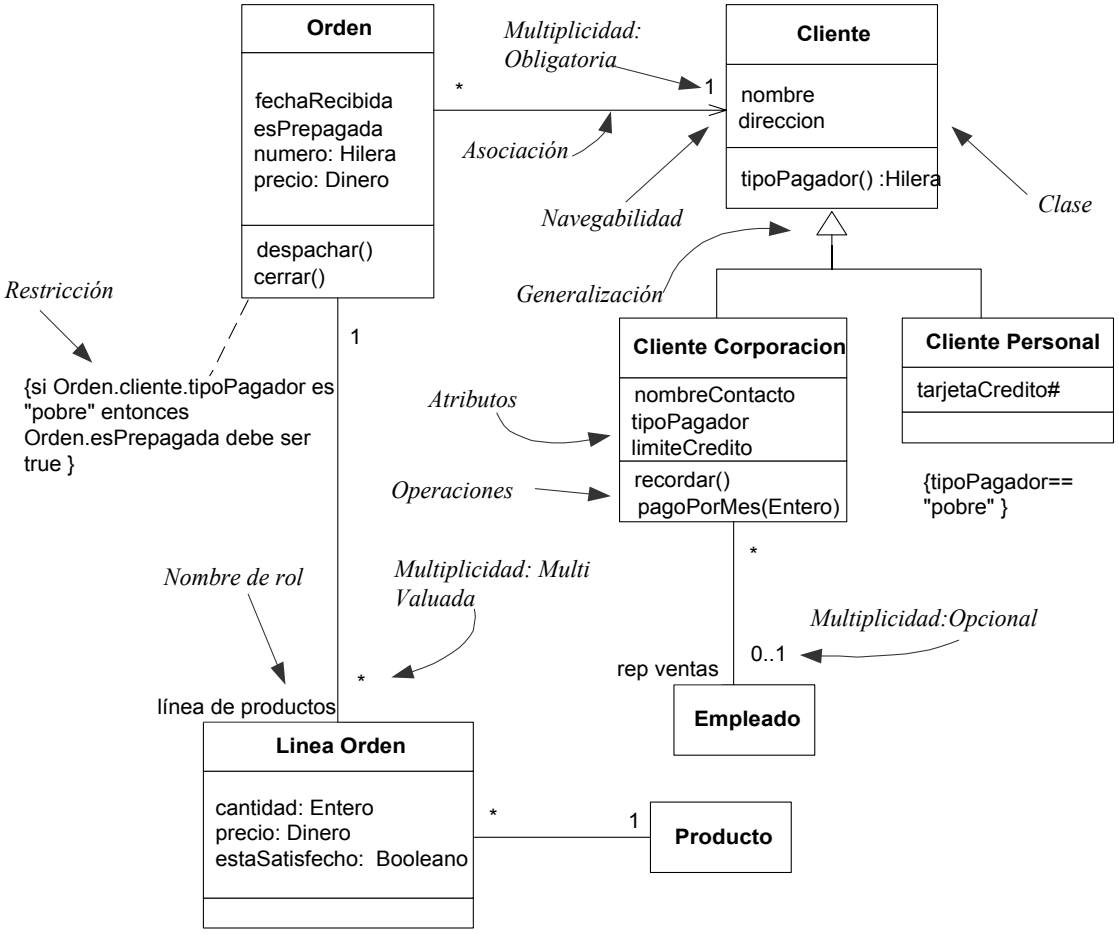


fig. 16 Diagrama de clases

Concepto	Descripción	Fig.
Refinamiento	Relación entre dos descripciones de la misma cosa, pero con diferente nivel de detalle. Por ejemplo, entre un tipo y una clase, una clase en el análisis y la misma clase en el diseño.	18 22
Calificador	Concepto que especifica cómo un objeto, de entre muchos, se identifica en una asociación.	19
Clase Asociación	Una clase conectada a una asociación. Puede tener atributos, operaciones y otras asociaciones.	19
Asociación Ternaria	Asociación a través de la cual se relacionan tres clases.	20
Asociación o Atributo derivado	Asociación o atributo cuyo valor se obtiene a partir de otras asociaciones o atributos.	21
Clase paramétrica	Denota una familia de clases cuya estructura y comportamiento están definidas independientemente de sus parámetros formales de clase.	22
Instanciación de	Consiste en hacer corresponder parámetros formales con parámetros	22

Clase paramétrica	reales para formar una clase concreta a partir de una clase paramétrica.	
Dependencia	Relación en la que una entidad (clase, caso de uso, paquete, componente, asociación, etc.) usa/depende de otra.	23
Interfaz	Conjunto de operaciones abstractas que especifican el comportamiento que una entidad (clase, componente, paquete) soportará al implementarlas.	23
Clase abstracta	Clase que no posee instancias.	24
Visibilidad	Caracterización de los componentes de una clase que especifica si pueden ser usados desde otras clases: + public: al componente se puede acceder desde cualquier otra clase. #protected: el componente puede ser accedido por los descendientes de la clase. -private: al componente se puede acceder sólo dentro de la clase que lo define.	

Existen otros conceptos que persiguen extender la semántica de algunos conceptos ya definidos.

Concepto	Descripción	Fig.
Etiqueta	Es una extensión de las propiedades de un elemento de UML, lo que permite crear nueva información en la especificación de ese elemento. Una etiqueta se presenta como una hilera entre llaves debajo del nombre del elemento que extiende.	24
Restricción	Condición semántica entre elementos de los modelos que especifican condiciones y proposiciones que deben ser verdaderas. Se presenta como una hilera entre llaves cerca del elemento asociado.	25
Estereotipo	Mecanismo que UML utiliza para extender la semántica pero no la estructura de elementos preexistentes. La interpretación semántica se da fuera de UML. Existe un conjunto de estereotipos predefinidos para las clases, entre los más importantes, tenemos: **Frontera**: especializa el uso de la clase para manipulación de entradas y presentación de salidas. Típicamente son ventanas, cajas de diálogos o clases de comunicación, como por ejemplo, las que implementan protocolos TCP/IP. **Entidad**: se utiliza para modelar objetos del negocio, tales como débito, facturas, contrato de seguro, etc. Típicamente son persistentes. **Control**: es utilizado para conectar objetos frontera con objetos entidad y manejar secuencias de operaciones en el interior del sistema. Típicamente la ejecución de cada caso de uso complejo queda asignada a un objeto de control responsable de coordinar las actividades del caso de uso.	26

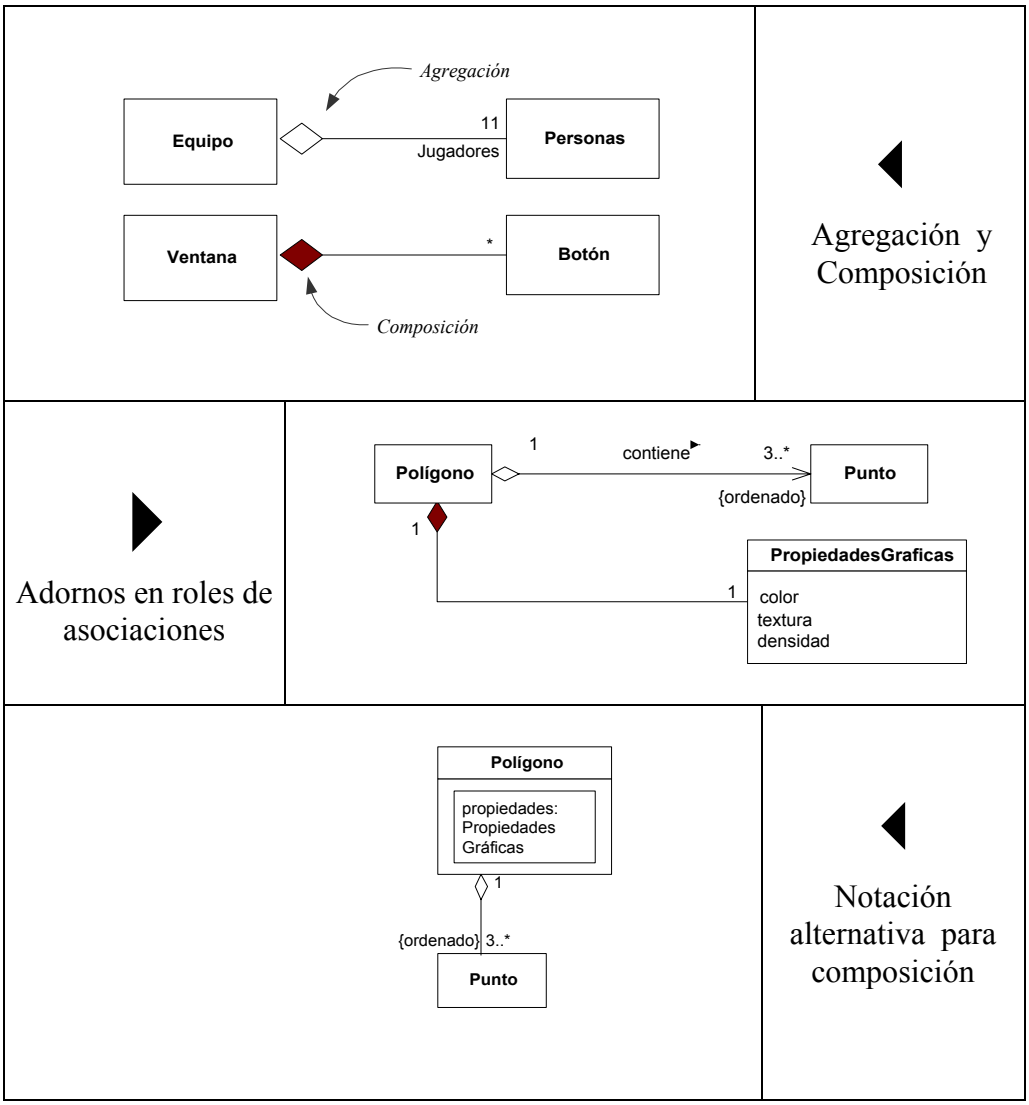


fig. 17 Representación de agregación y composición

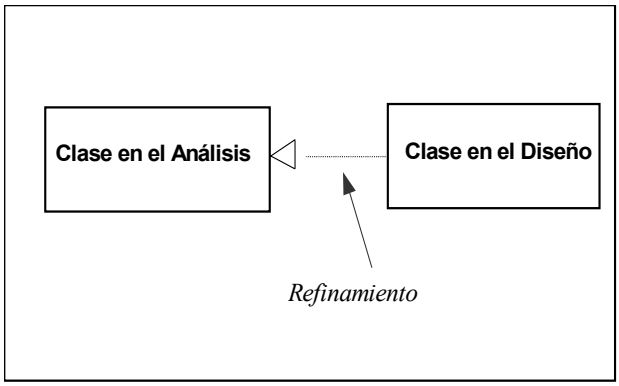


fig. 18 Representación de refinamiento

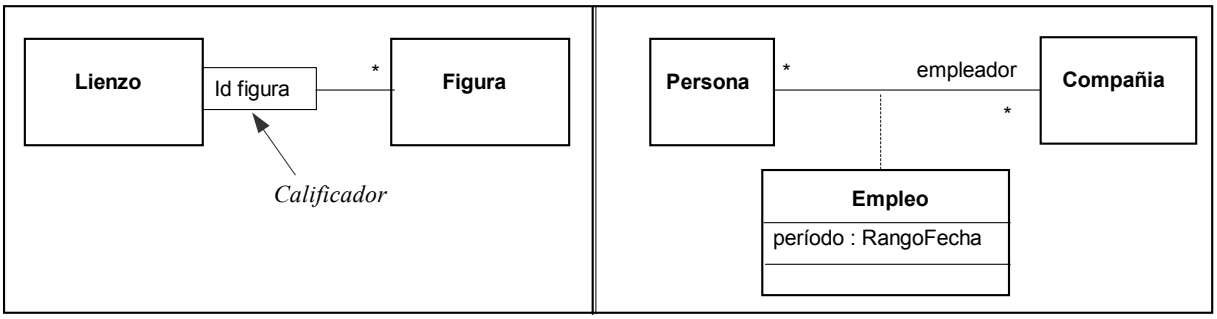


fig. 19 Asociación calificada y clase asociación

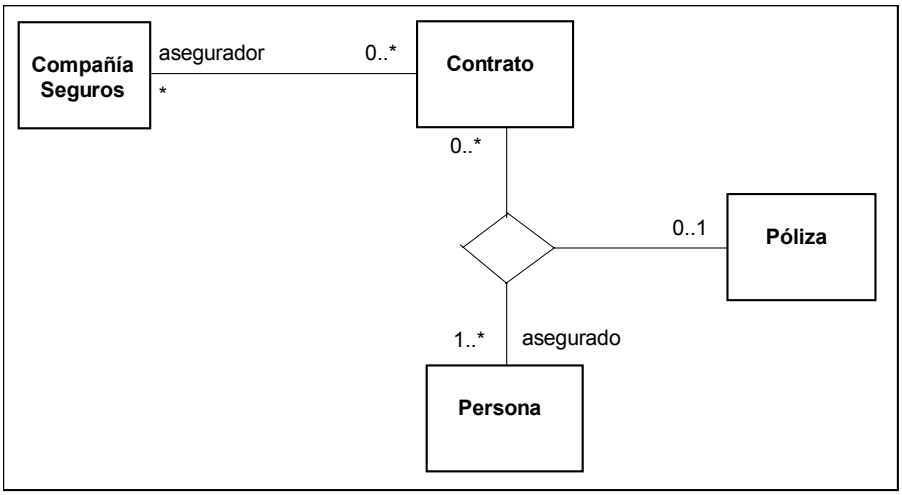


fig. 20 Asociación ternaria

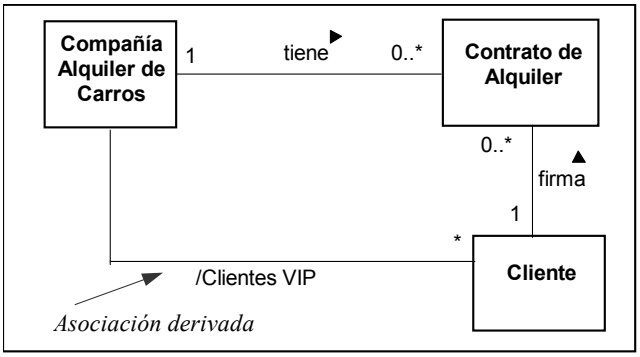


fig. 21 Asociación derivada

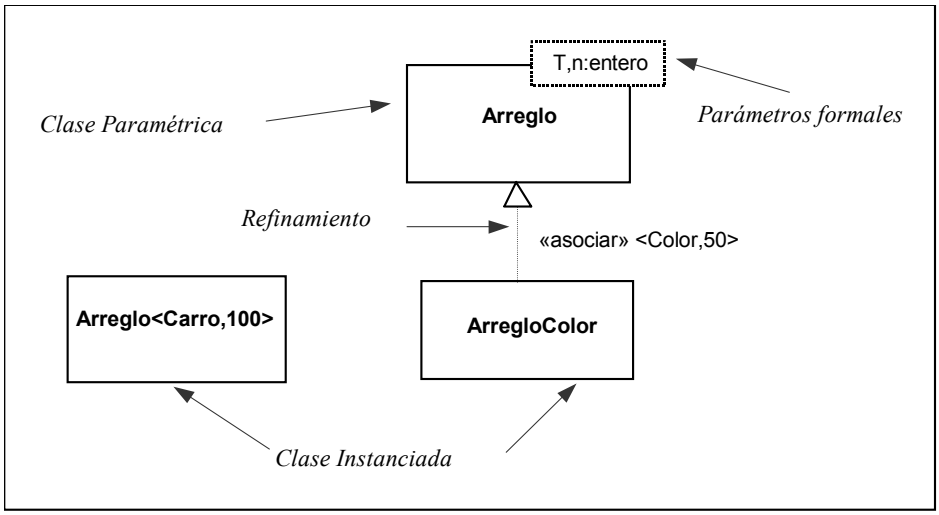


fig. 22 Clase paramétrica Arreglo con los parámetros T (una clase) y n (un entero). Dos formas de representar instanciación

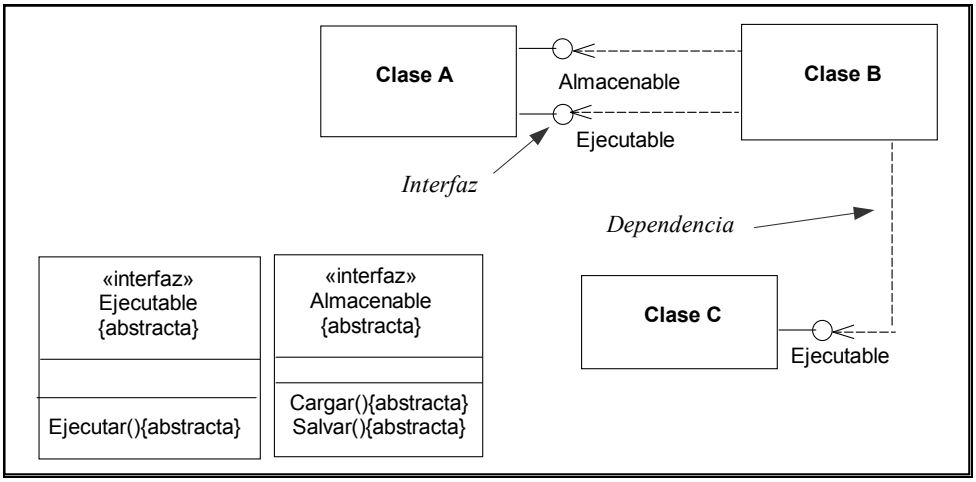
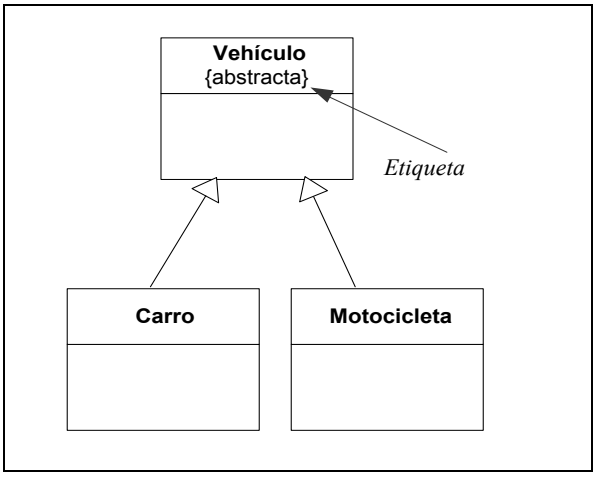


fig. 23 Representación de interfaz

fig. 24 Clase abstracta



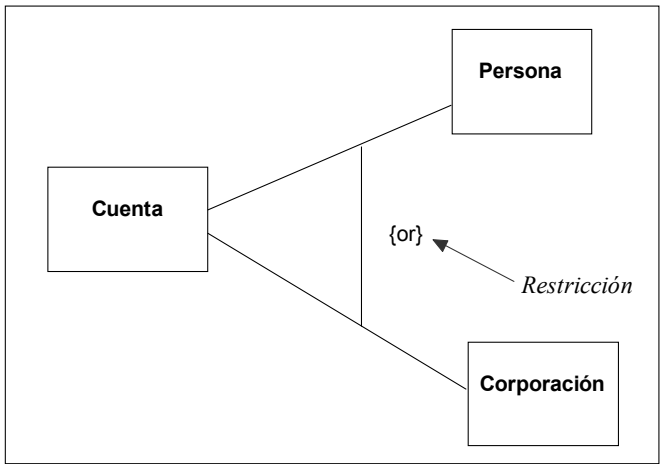


fig. 25 Restricción de asociación (asociaciones excluyentes)

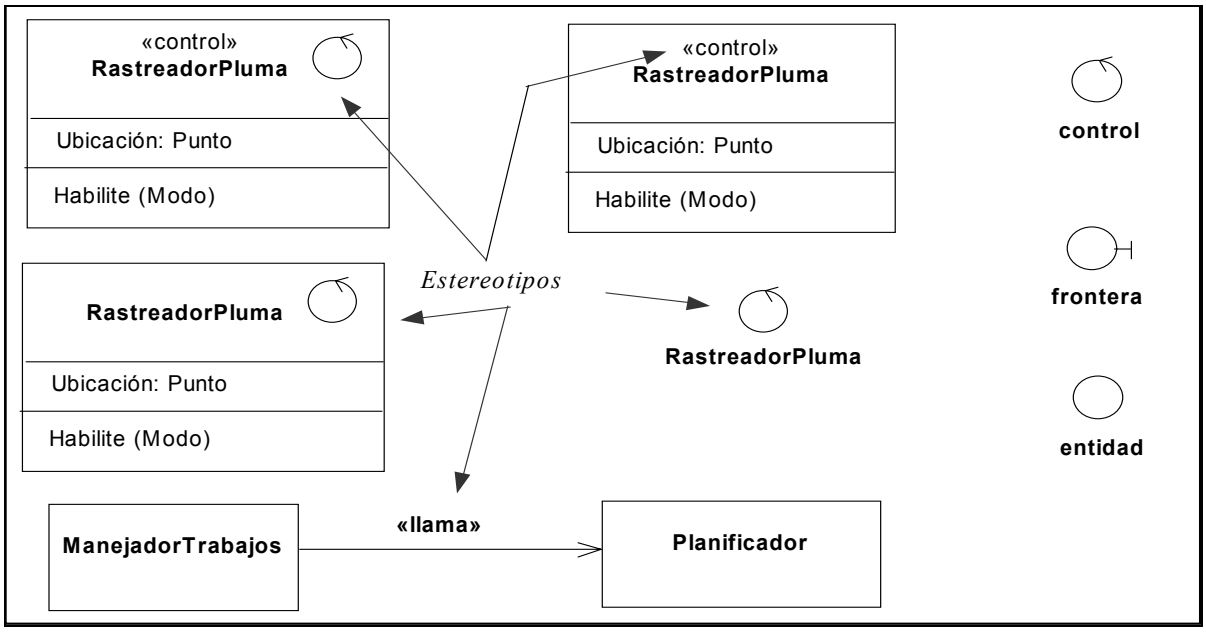


fig. 26 Diversas maneras de presentar estereotipos

3.2 Diagrama de Objetos

Mientras que un diagrama de clases presenta las clases y sus relaciones, un diagrama de objetos presenta instancias específicas de esas clases y los enlaces específicos entre esas instancias en un momento determinado (fig. 27)

Concepto	Descripción	Fig.
Objeto	Instancia de una clase.	27 28
Enlace	Instancia de asociación.	27
Objeto compuesto	Objeto que incorpora en su estado otros objetos.	29

Aunque un diagrama de objetos permite ilustrar cómo un diagrama de clases complejo puede ser instanciado en objetos, usualmente se prefiere emplear los diagramas de colaboración, ya que además de presentar la estructura estática de los objetos afectados, sus relaciones relevantes así como atributos y operaciones, incluye también una descripción de la secuencia de mensajes intercambiados por los objetos en el momento que se produce la interacción.

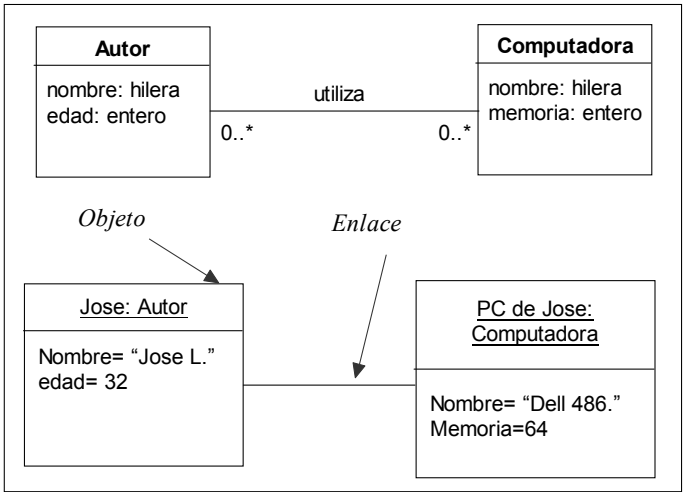


fig. 27 Diagrama de clases y diagrama de objetos

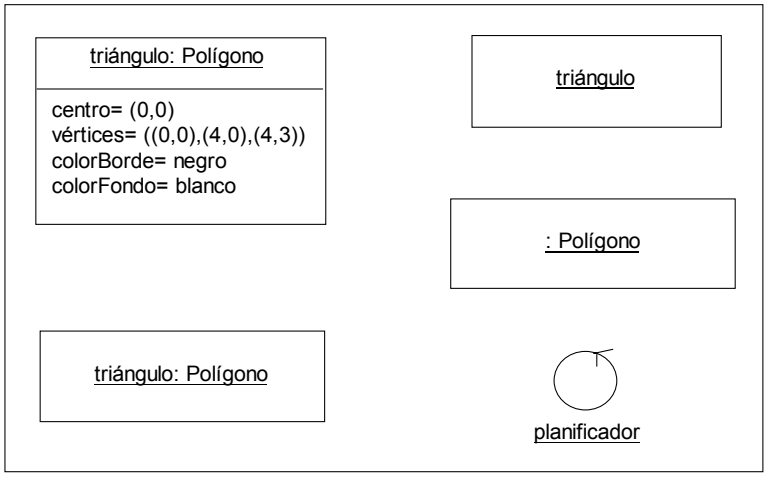


fig. 28 Representaciones de objeto

En la fig. 28 se muestra un conjunto de representaciones para los objetos. Un objeto puede ser nombrado, es decir, se tiene la instancia específica de una clase, la que es identificada por su nombre (como el caso del objeto **triángulo** que pertenece a la clase **Polígono**, o el objeto de control **planificador**). Pueden existir objetos anónimos, los cuales representan cualquier instancia de una clase en particular y no una instancia específica (se indican con el nombre de la clase, subrayada y antecedida por dos puntos).

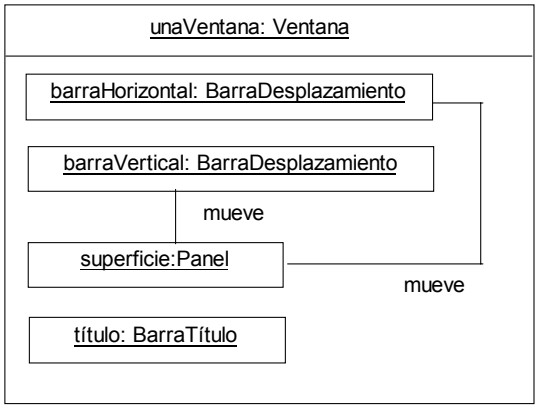


fig. 29 Objeto compuesto

En la fig. 29 se presenta un objeto nombrado (**unaVentana)**, compuesto por otros objetos que a su vez han sido nombrados (**barraHorizontal, barraVertical, superficie, título**).

4 Elementos para modelar la vista de comportamiento (dinámica)

Un diagrama de clases refleja la estructura colaborativa de tipos de objetos ideada para lograr comportamientos que vienen a satisfacer los requerimientos de los usuarios. Bajo estas estructuras se puede producir una dinámica de comportamiento de los entes involucrados, objetos interactuando entre sí, pasándose mensajes en una y otra dirección, eventos sucediéndose constantemente, solicitudes que van, resultados que vienen, objetos se crean, se destruyen, se pasan como parte del flujo de información, en fin, una dinámica que representa las tareas necesarias para proveer los comportamientos del sistema.

Desde la perspectiva del comportamiento, UML ofrece elementos para proyectar en el modelo dinámico dos aspectos que son de gran interés:

- La descripción de cómo colaboran los grupos de objetos para lograr un comportamiento específico.
- La descripción de la evolución dinámica del sistema.

En el primer caso tenemos diagramas de secuencia y diagramas de colaboración, con los que se representan interacciones que describen el comportamiento externo de los objetos y cómo estos objetos interactúan unos con otros. Normalmente estos diagramas captan el comportamiento de un caso de uso, presentando ejemplos de los objetos y los mensajes que se transmiten.

En el segundo caso disponemos de diagramas de transición de estados y diagramas de actividad, con los que podemos representar, por una parte, cómo los objetos reaccionan a los eventos y cómo alteran sus estados internos y por otra, el trabajo que desempeñan los objetos en términos de las actividades realizadas.

UML no ofrece facilidades propias para describir el efecto de una operación sobre el estado del sistema. El lenguaje OCL (Object Constraint Language – lenguaje para restricción de objetos) puede ser utilizado para describir operaciones con base en predicados.

4.1 Diagrama de Secuencia

Este tipo de diagrama muestra una interacción ordenada en una secuencia temporal, presentando los objetos que participan y los mensajes que ellos intercambian, arreglados en un orden respecto del tiempo. No se muestran las relaciones entre los objetos.

Concepto	Descripción	Fig.
Objeto	Entidad que envía o recibe el mensaje.	30
Mensaje	Solicitud enviada a un objeto para que lleve a cabo alguna operación.	30
Mensaje simple	Mensaje que presenta cómo el control es pasado de un objeto a otro sin describir detalles de la comunicación.	30
Mensaje sincrónico	Mensaje que presenta el flujo de control anidado, implementado como llamada a una operación.	30

Mensaje asincrónico	Mensaje que presenta el flujo de control asincrónico, donde no hay un retorno explícito. El que envió el mensaje continúa ejecutando sin necesidad de esperar el retorno.	31
Línea vital	Representa la vida del objeto durante la interacción.	30
Retorno	Mensaje simple que representa el retorno de un mensaje y no un nuevo mensaje.	30
Activación	Estado en que el objeto es capaz de enviar o recibir mensajes para su procesamiento.	30 31
Autodelegación	Envío de un mensaje por parte de un objeto a sí mismo.	30
Creación	Aparición de un objeto durante la interacción.	30
Destrucción	Desaparición de un objeto durante la interacción.	30
Condición	Expresión Booleana asociada a un mensaje. Debe ser verdadera para que el mensaje pueda ser enviado.	30
Iteración	Indica que un mensaje es enviado a múltiples objetos receptores.	30

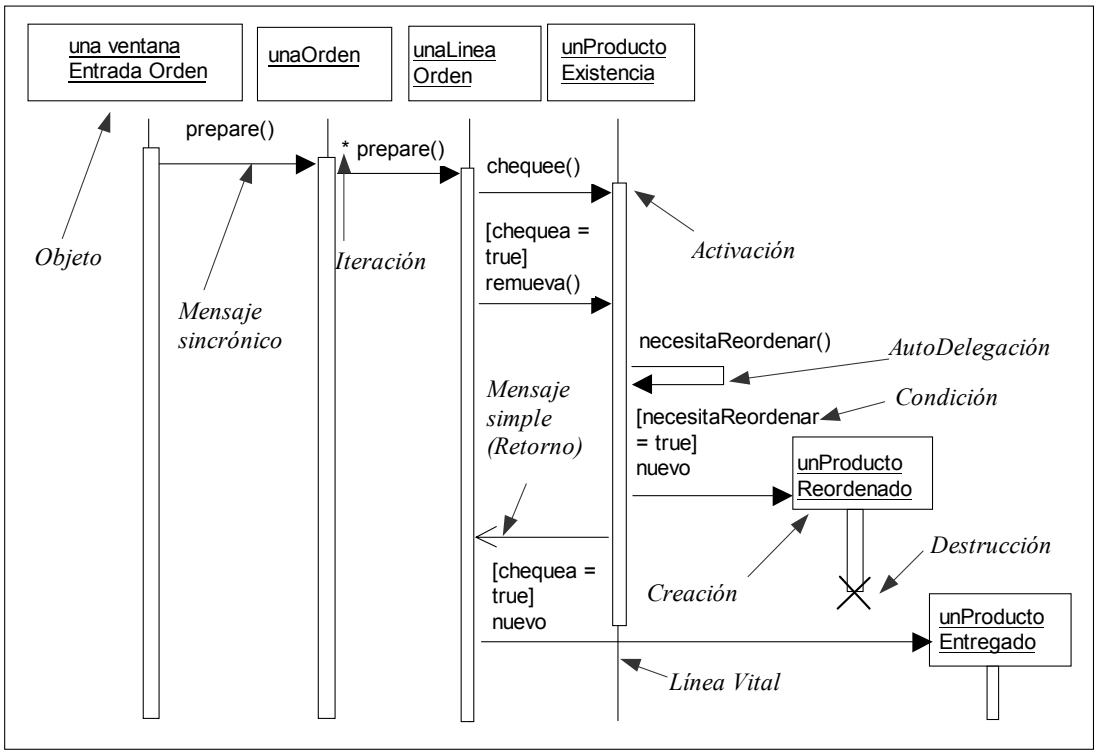


fig. 30 Diagrama de secuencia

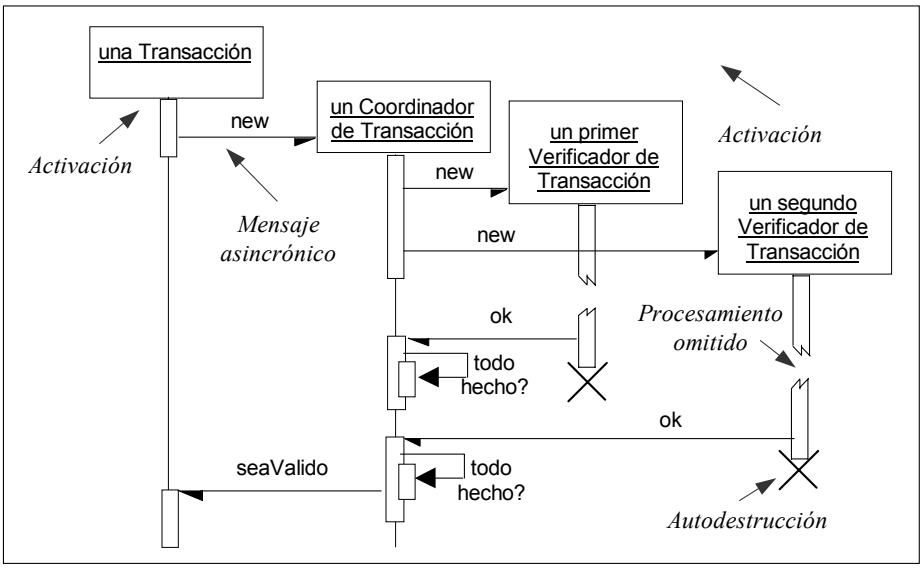


fig. 31 Procesos concurrentes y activación

4.2 Diagrama de Colaboración

Un diagrama de colaboración es otra manera de representar la interacción entre los objetos. Al igual que en el diagrama anterior se muestran los objetos y los mensajes que se intercambian, pero arreglados espacialmente.

Se incluyen los enlaces entre las instancias, que vienen a ser las vías de comunicación por donde circulan los mensajes. Puesto que el tiempo ha desaparecido como dimensión, se establece un ordenamiento temporal mediante números de secuencia.

Concepto	Descripción	Fig.
Enlace	Instancia de una asociación.	32
Número de Secuencia	Número que establece un ordenamiento temporal en el envío de mensajes, lo cual permite apreciar la secuencia ocurrida.	32
Restricción estándar de enlace	Extensión del UML que establece una restricción sobre el enlace. Indica los diversos tipos de implementación. Estas restricciones son: • Global: Instancia visible en el sistema debido a su alcance global. • Local: Instancia visible como local dentro de una operación. • Parámetro: Instancia visible como parámetro en una operación. • Auto: Especifica que un objeto se envía mensajes a sí mismo.	34
Marcador de creación/destrucción	Restricción que marca un objeto o enlace durante una interacción dependiendo de la existencia de este. Nuevo: indica que ha sido creado durante la interacción en cuestión. Destruido: indica que ha sido destruido durante la interacción. Transiente: indica que es creado y destruido durante la interacción.	34

Objeto Activo	Objeto que posee su propio hilo de control y puede iniciar una actividad. Tiene la capacidad de ejecutar concurrentemente.	34

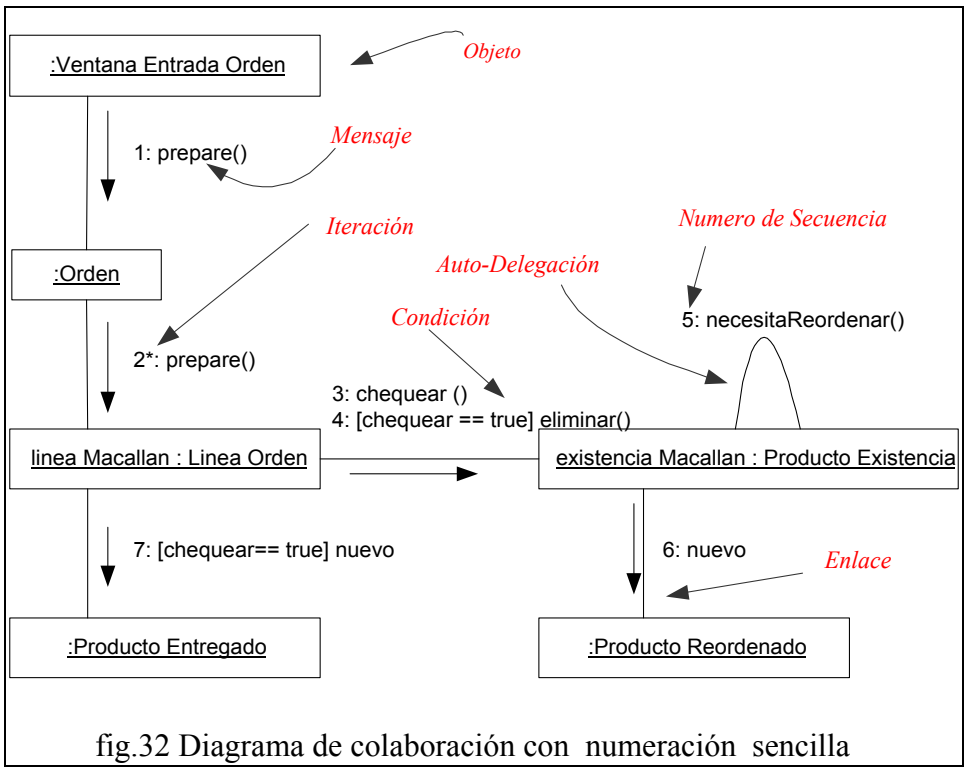


fig.32 Diagrama de colaboración con numeración sencilla

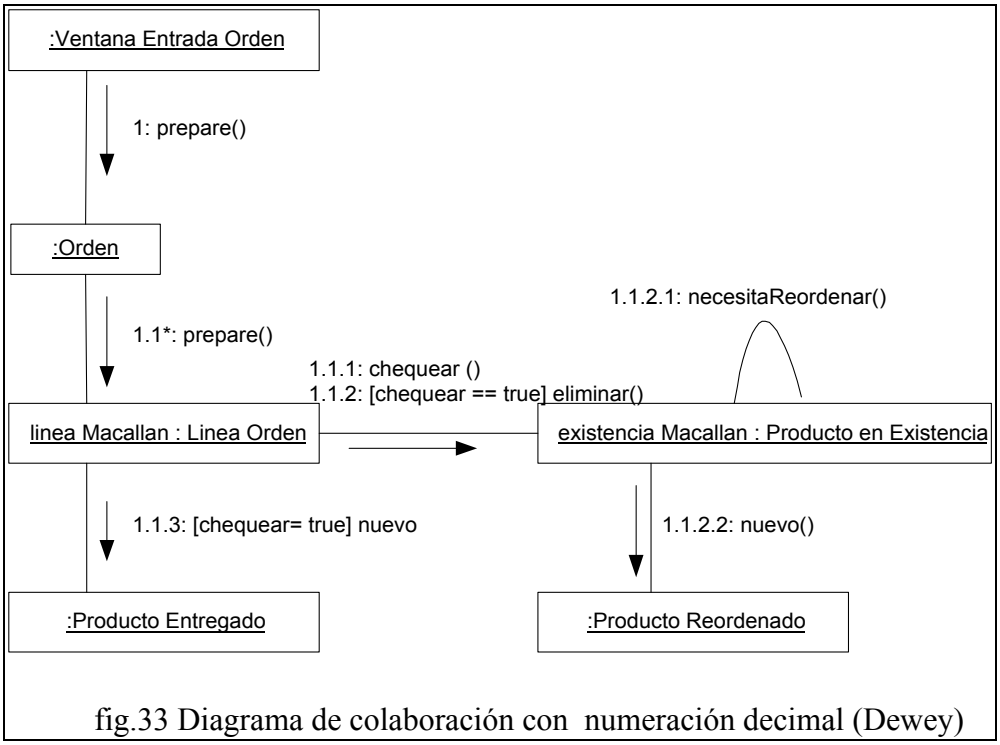


fig.33 Diagrama de colaboración con numeración decimal (Dewey)

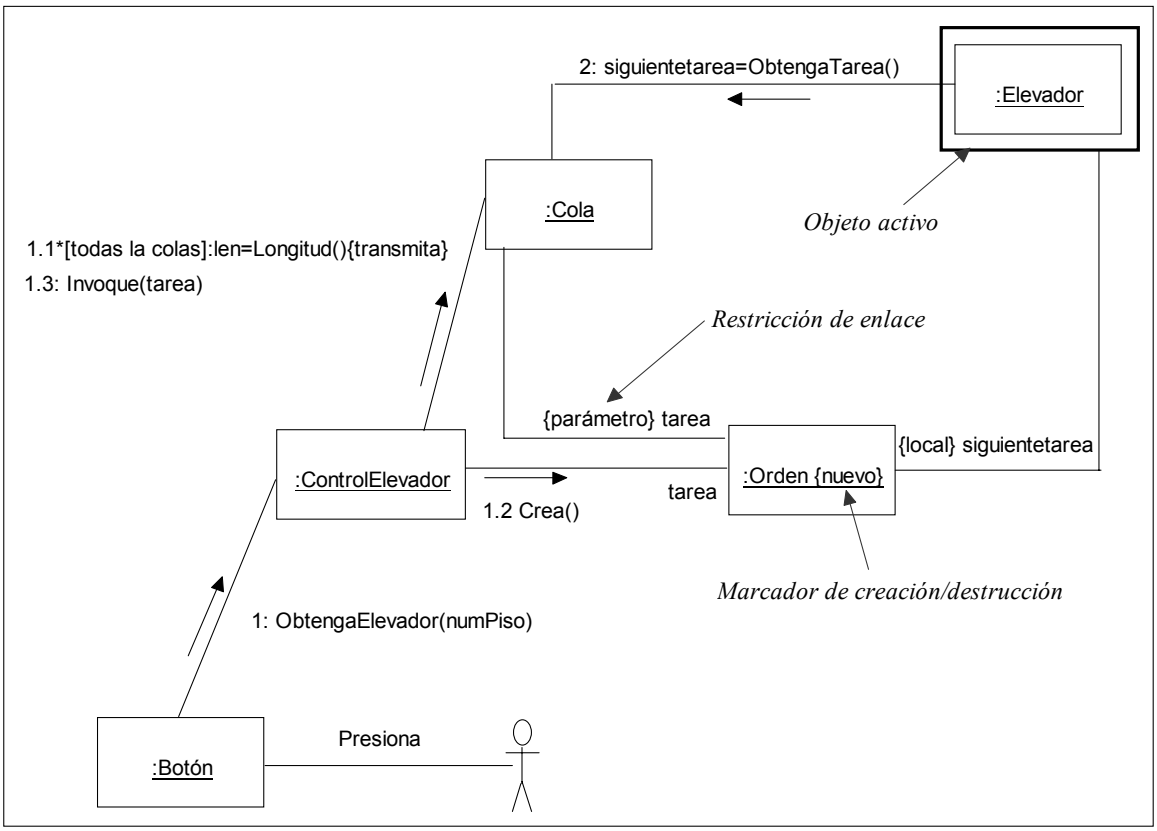


fig. 34 Colaboración donde un actor presiona un botón para obtener un elevador a su piso

4.3 Diagrama de Transición de Estados

Los diagramas de transición de estados se utilizan para mostrar el espacio de estados de una clase determinada, los eventos que provocan una transición de un estado a otro y las acciones que resultan de ese cambio de estado.

Los principales conceptos que se utilizan para construir estos diagramas se definen a continuación.

Concepto	Descripción	Fig.
Estado	Resultado de actividades previas realizadas por el objeto, y determinada por los valores de sus atributos y enlaces a otros objetos.	35
Evento	Una ocurrencia significativa en el tiempo o en el espacio (un suceso).	35
Transición de Estados	Cambio de estado provocado por la ocurrencia de un evento.	35

Estado inicial	Estado de partida por omisión para el procesamiento en estudio.	35
Estado final	Estado donde ha finalizado el procesamiento en estudio.	38

Existen otros conceptos que se clasifican como avanzados los cuales se emplean para realizar el modelaje de sistemas que implican mayor complejidad.

Concepto	Descripción	Fig.
Acción	Proceso asociado a las transiciones de estados, se considera que ocurre rápidamente y no es interrumpible.	35
Actividad	Proceso asociado al estado. Puede durar un cierto tiempo y ser interrumpible.	35
Condición de resguardo	Expresión Booleana asociada a la transición de estados. Tiene que ser verdadera para que la transición se pueda disparar.	35
Cláusula enviar	Caso especial de una acción. Permite enviar un mensaje durante la transición entre dos estados.	38
Indicador de Historia	Indicador utilizado para memorizar estados internos.	38
Subestados and	Subestados concurrentes, pueden estar activos al mismo tiempo.	39
Subestados or	Subestados que no pueden estar activos al mismo tiempo.	39

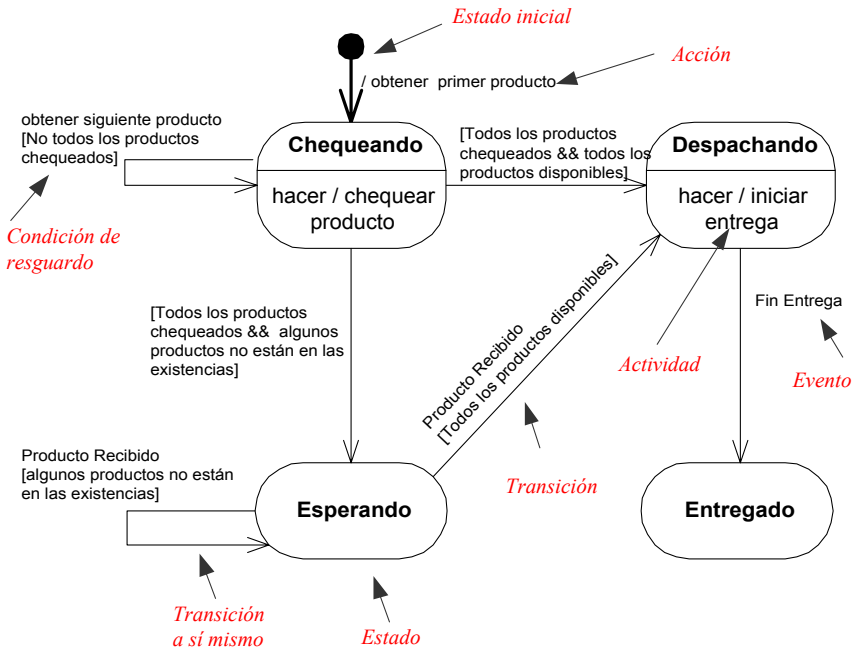


fig. 35 Diagrama de estados

En la fig.36 se agrega el estado cancelado y las transiciones cancelar que van hacia él. En la fig.37 se introduce un superestado (Activo) que comprende los subestados: Chequeando, Despachando y Esperando. Ahora la presentación es más simple, al mostrar la transición hacia el estado Cancelado, la cual se da desde cualquiera de los subestados de Activo.

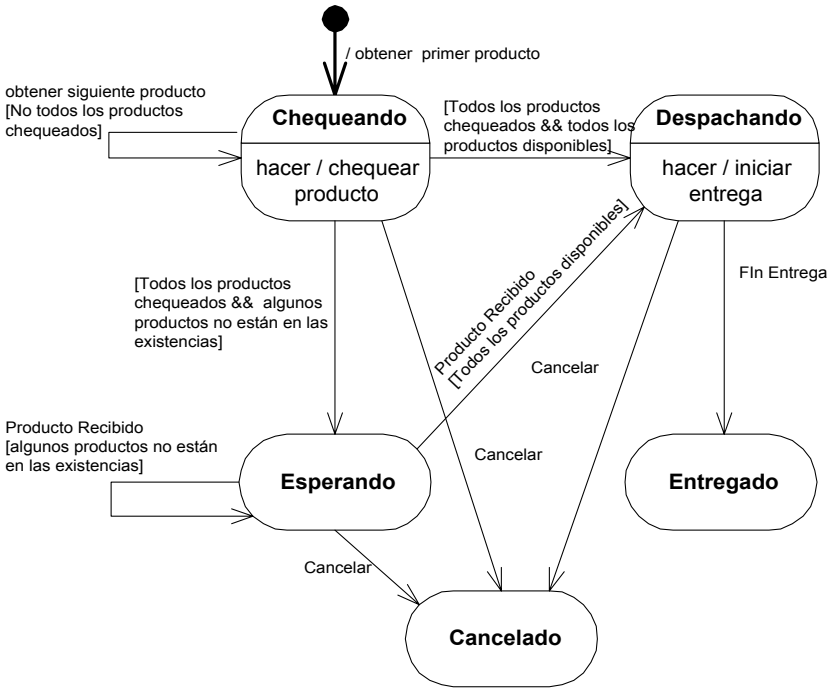


fig. 36 Diagrama de estados sin superestados

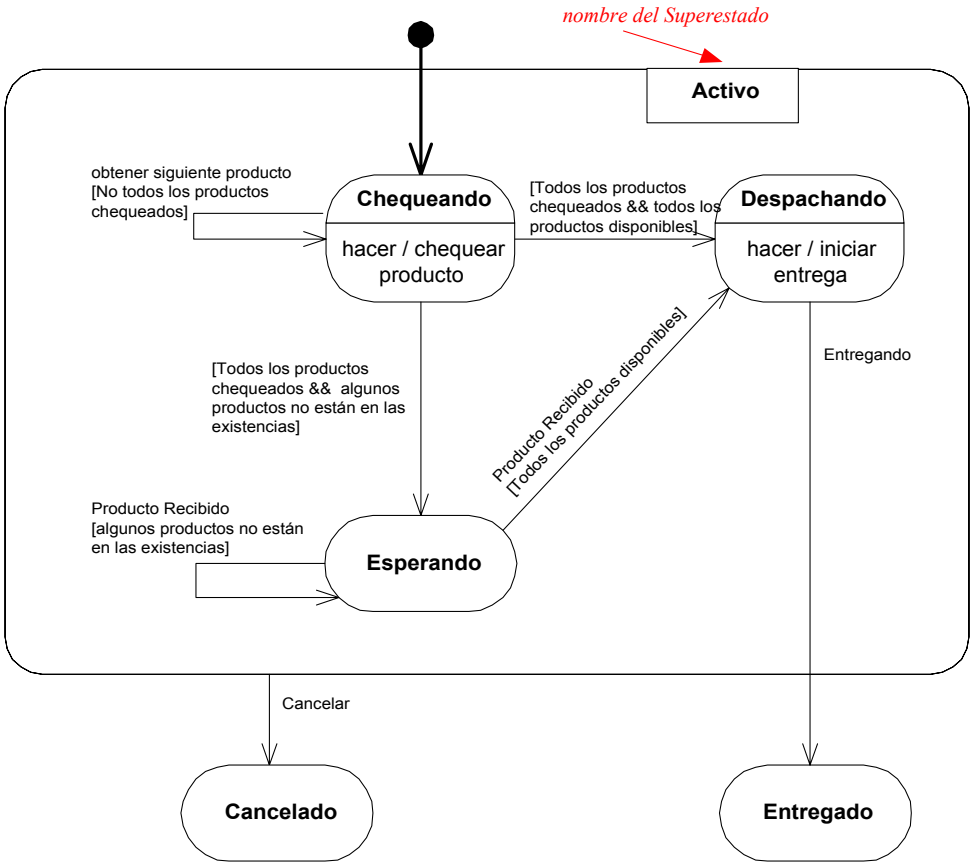


fig 37 Diagrama de estados con superestados

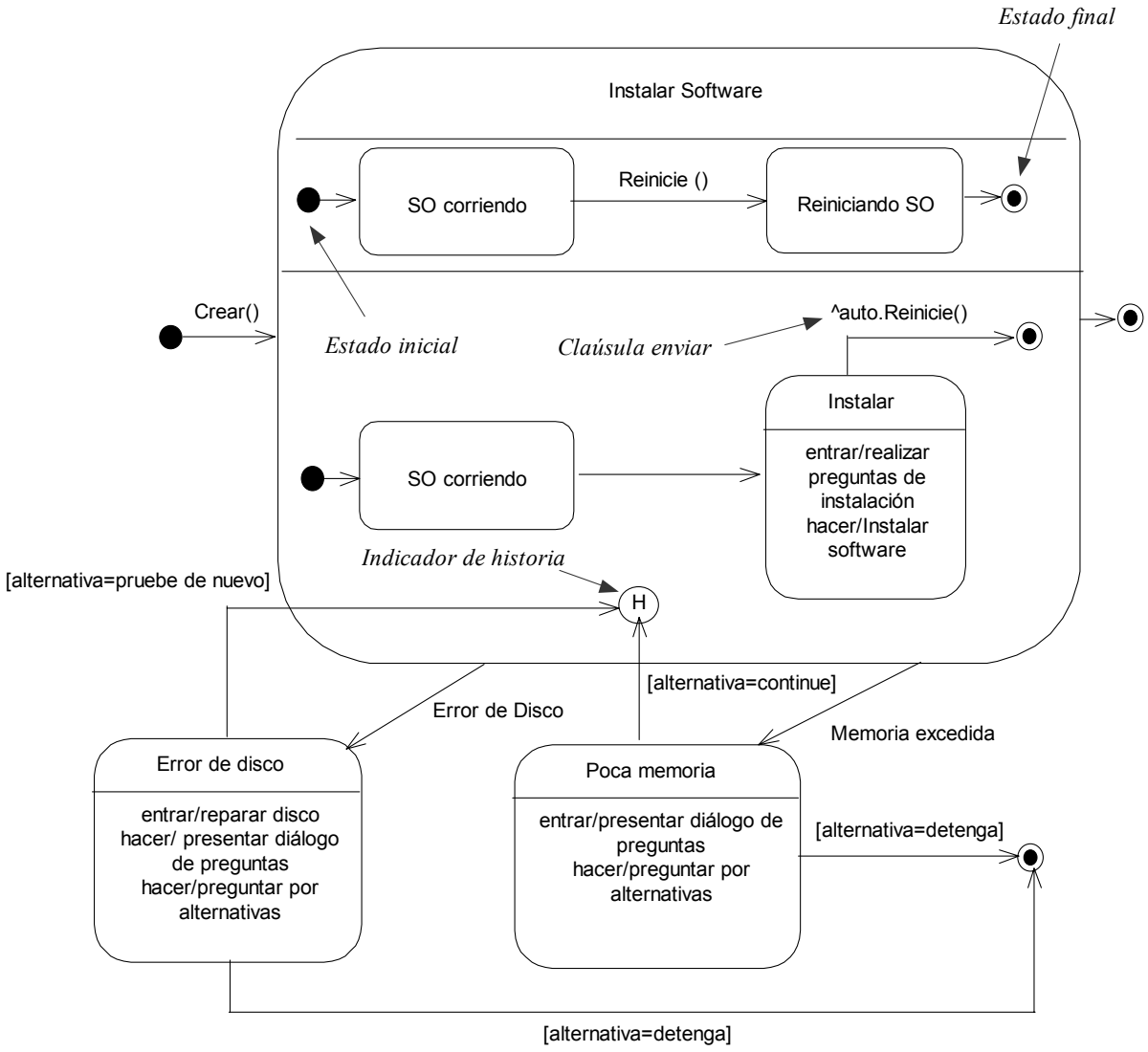


fig. 38 Diagrama de estados con múltiples componentes

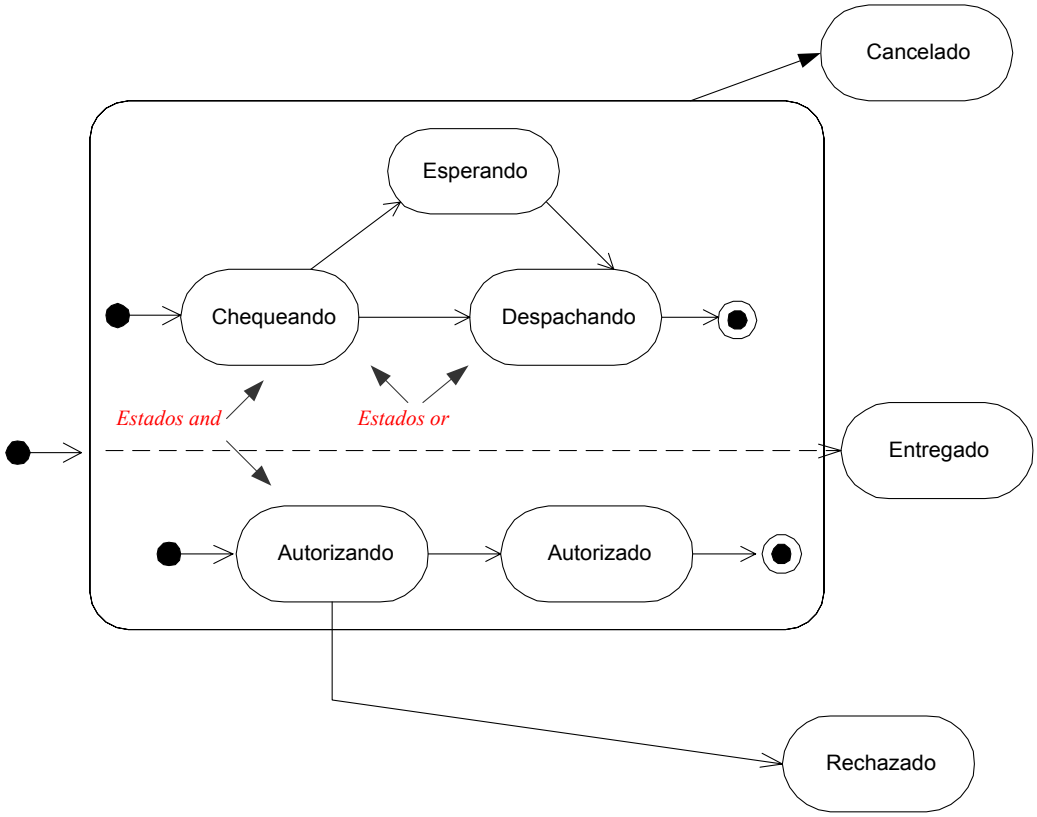


fig. 39 Diagrama de estados concurrentes

4.4 Diagrama de Actividad

Este diagrama ayuda a mostrar las acciones y resultados que ocurren ya sea en todo el modelo que se está creando, en un caso de uso, en una clase o en una operación. Resulta ser una variante de los diagramas de estados. Los estados (denominados estados-acción) representan acciones y las transiciones se disparan cuando las acciones en el estado han concluido.

El propósito principal es describir flujos debidos al procesamiento interno y no como respuesta a eventos externos. Los diagramas presentan algunas características:

- Un símbolo en forma de diamante es utilizado para presentar puntos de decisión.
- Pueden manejar procesos paralelos
- Las acciones se pueden colocar en carriles para ver quién es el responsable de ellas o dónde residen en la organización.

Concepto	Descripción	Fig.
Actividad (Acción)	Tarea que necesita ser llevada a cabo.	40
Transición	Cambio de una acción/actividad a otra, producto de la conclusión de la acción/actividad previa.	40
Barra de Sincronización	Lugar donde llega una transición y se divide en varias ramas que pueden ejecutarse concurrentemente (difusión) o donde llegan varias transiciones que provienen de actividades concurrentes (reunión).	40
Condición de resguardo	Expresión Booleana asociada a la transición. Tiene que ser verdadera para que la transición se pueda disparar.	40
Actividad de decisión	Actividad donde dependiendo de una condición se pueden tomar diferentes cursos de acción.	40
Carriles	Mecanismo utilizado para agrupar actividades, normalmente con respecto de los responsables de ejecutarlas.	44

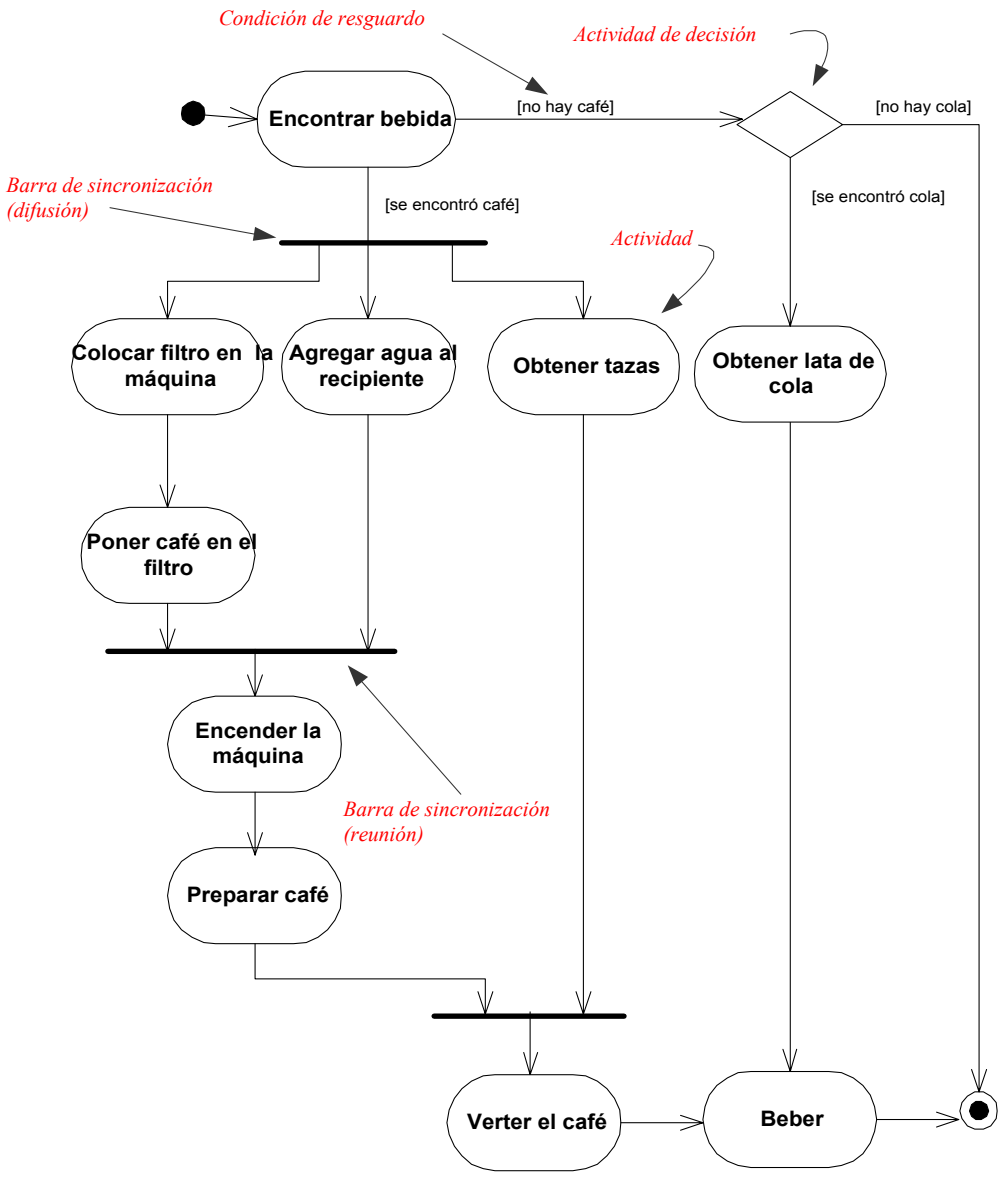


fig. 40 Diagrama de actividad

En las figuras siguientes (41, 42, 43) se presentan diversos diagramas de actividad ya sea en forma separada o en combinación, dependiendo de las actividades que resulten ser comunes.

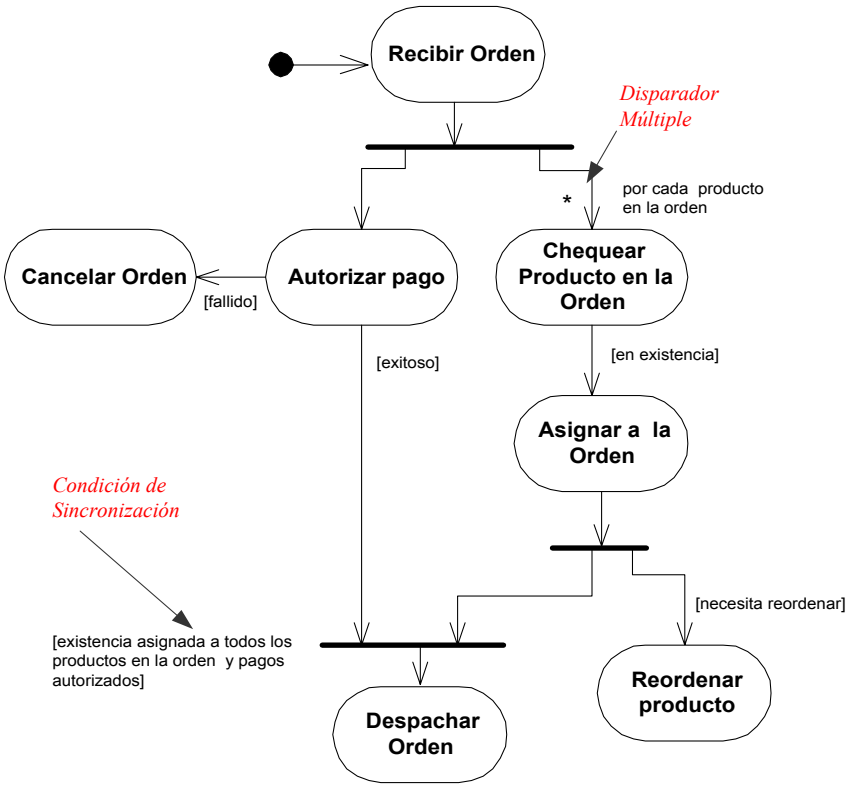


fig.41 Recepción de orden

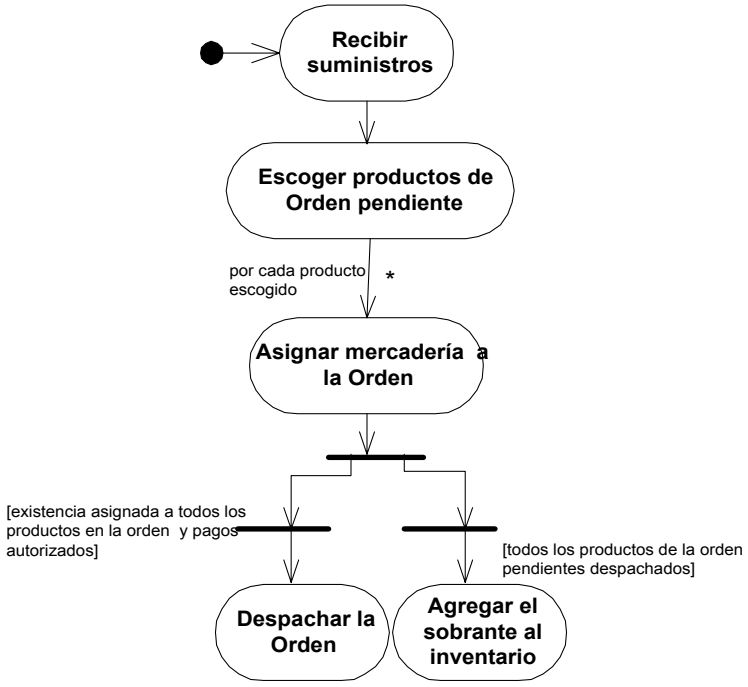


fig. 42 Recepción de suministros

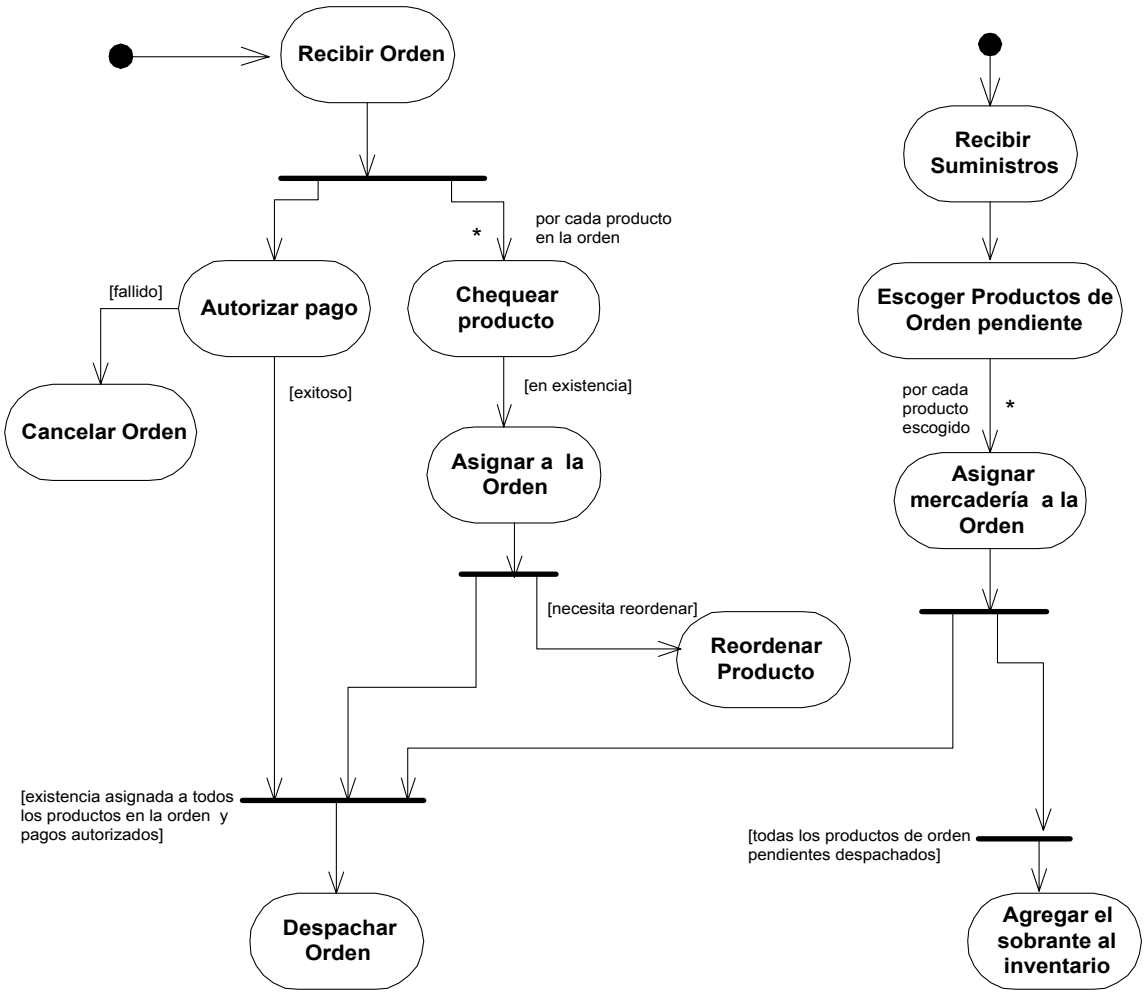


fig. 43 Combinación de recepción de orden y suministros

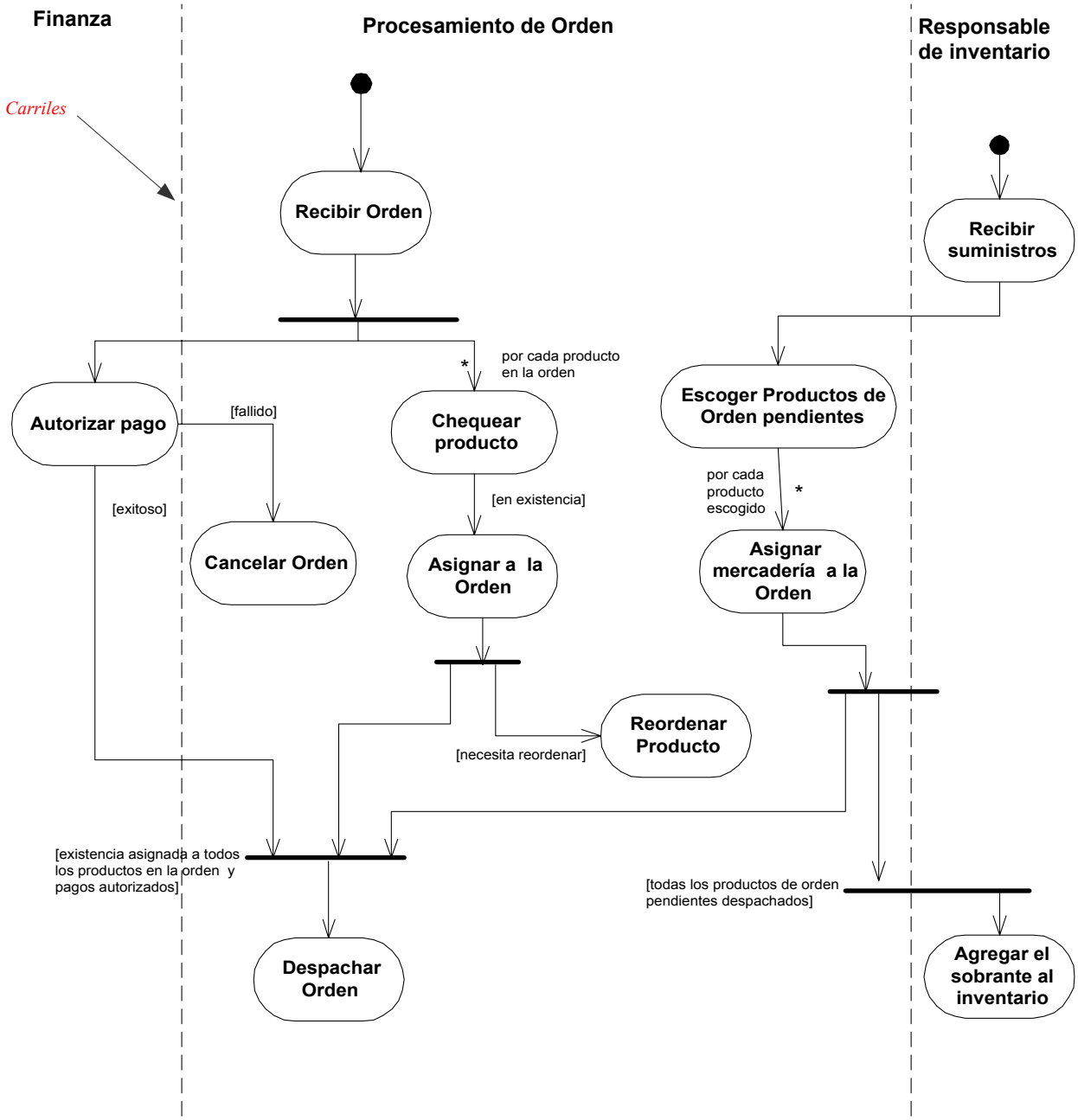


fig. 44 Diagrama de actividad con carriles

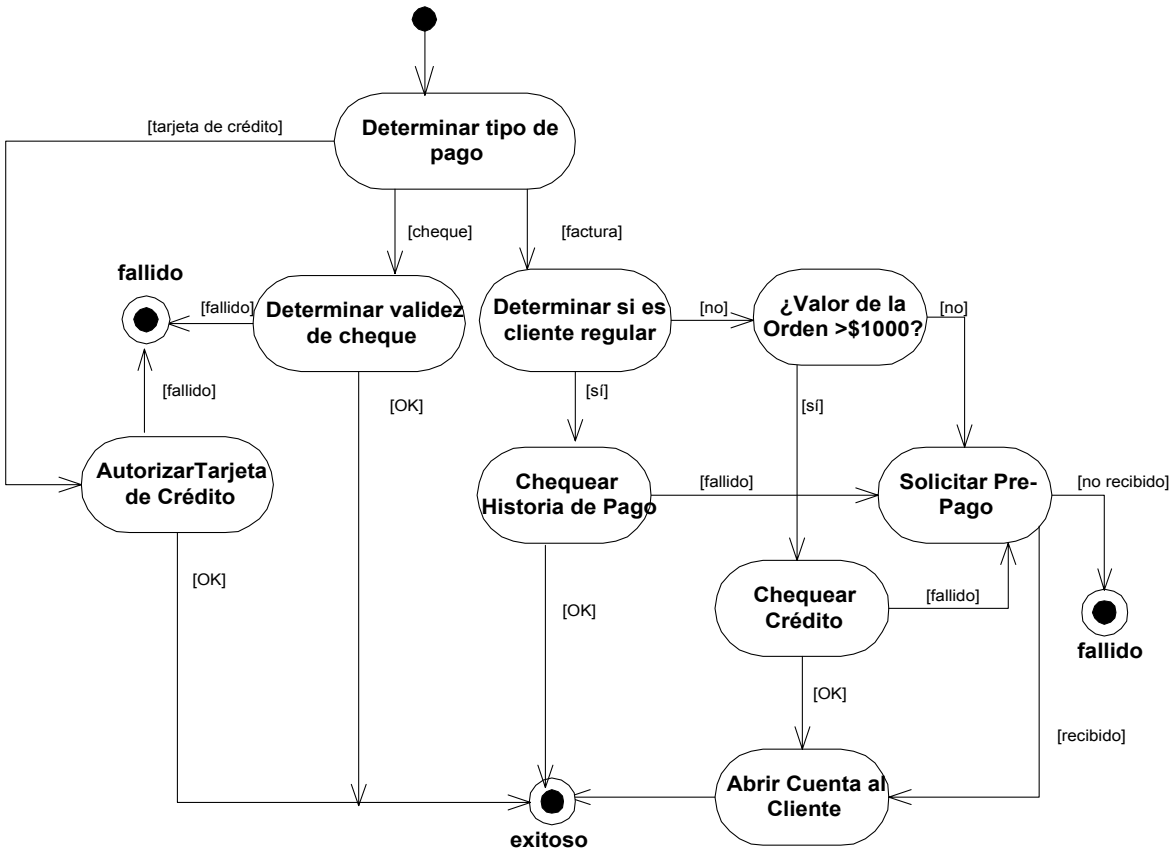


fig. 45 Descomposición de una actividad

Además de las acciones y sus transiciones, los diagramas de actividad pueden mostrar el flujo interno de objetos (entre corchetes se puede mostrar el estado de los objetos).

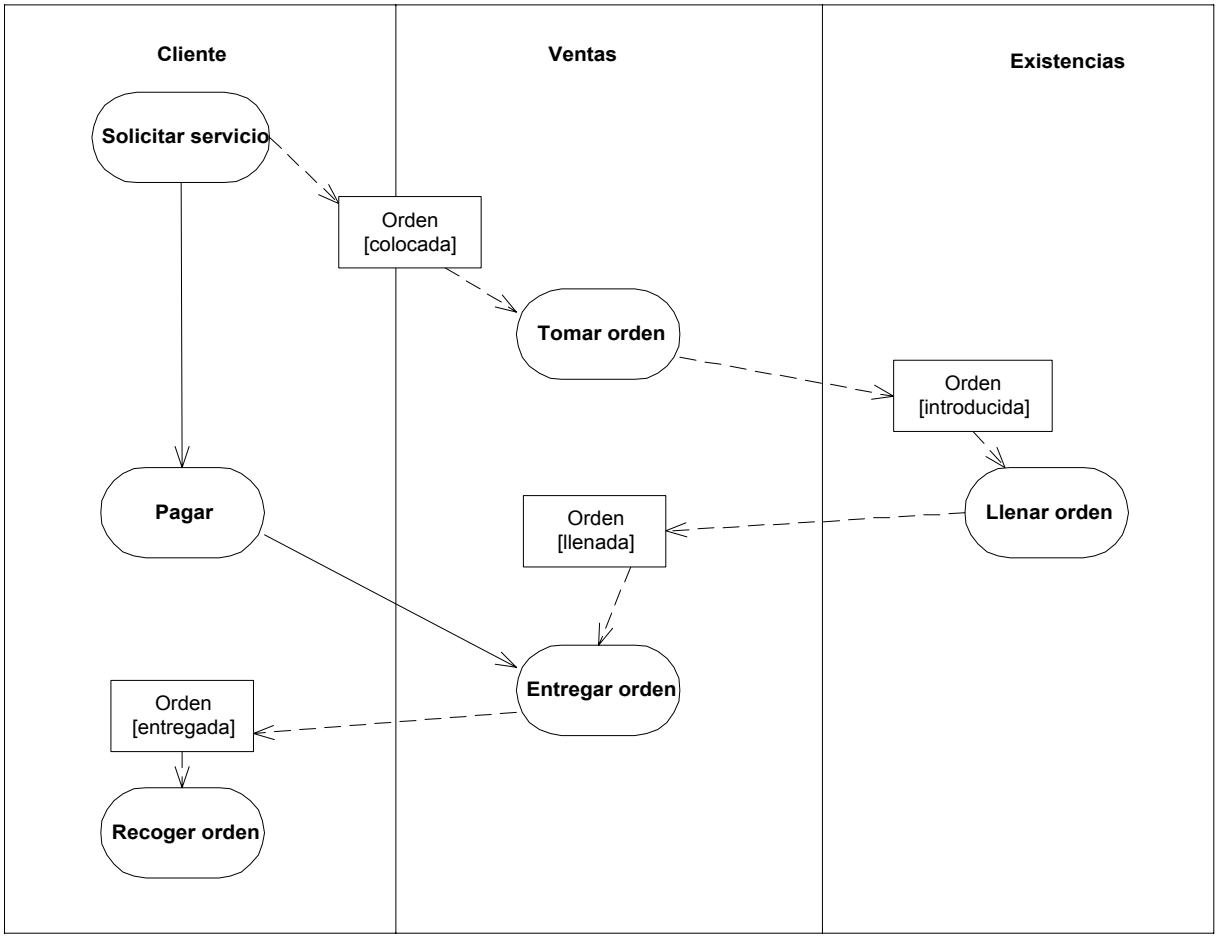


fig. 46 Acciones y flujo de objetos

5 Elementos para modelar la implementación

UML permite presentar aspectos de implementación, incluyendo la estructura de código fuente y la estructura de implementación en tiempo de ejecución. Para ello provee diagramas de componentes y diagramas de despliegue.

5.1 Diagrama de Componentes

Un diagrama de componentes describe los componentes de software y las relaciones de dependencias que se presentan entre ellos, representando la estructura del código. En este diagrama se presentan los componentes como una representación de tipos y no de instancias. Para presentar instancias de componentes se suele utilizar diagramas de despliegue.

Concepto	Descripción	Fig.
Componente	Implementación en la arquitectura física de los conceptos y la funcionalidad definida en la arquitectura lógica.	47
Fuente	Componente significativo en tiempo de compilación. Típicamente un archivo fuente que implementa una o más clases.	47
Binario	Componente típicamente código objeto que es el resultado de la compilación de un código fuente. Puede ser un archivo de código objeto, una biblioteca estática o una biblioteca dinámica.	47
Ejecutable	Componente que consiste de un programa ejecutable, resultado del enlace de los componentes binarios.	47
Dependencia	Conexión entre componentes, indicando que un componente necesita el otro componente para tener una definición completa.	47

5.2 Diagrama de Despliegue

Un diagrama de despliegue muestra la configuración de las unidades de ejecución (procesador, dispositivos) y componentes de software, procesos y objetos que residen en ellas. Tanto los elementos físicos como lógicos pueden representarse en el diagrama como tipos o como instancias (fig. 48). Un nodo que contiene un componente indica que el componente se ejecuta en ese nodo.

Concepto	Descripción	Fig.
Nodo	Son los objetos físicos que tienen algún tipo de recurso computacional. Incluye computadoras con procesadores así como dispositivos, tales como impresoras, lectores de tarjetas, etc.	49
Conexión	Presenta la ruta de comunicación a través de la cual el sistema interactúa.	49

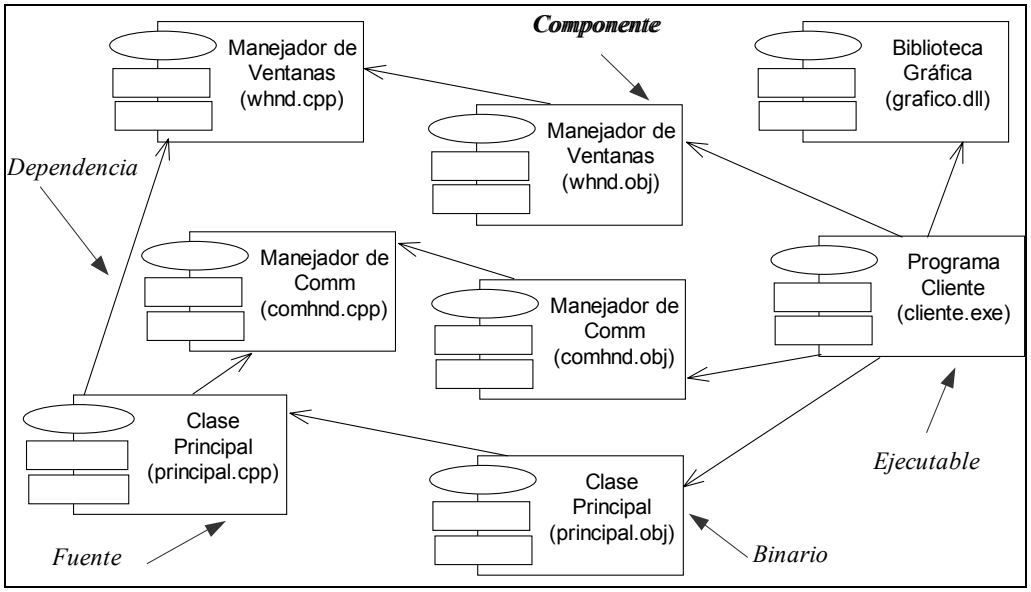


fig. 47 Diagrama de componentes que presenta varios componentes (fuente, binario, ejecutable) y sus dependencias

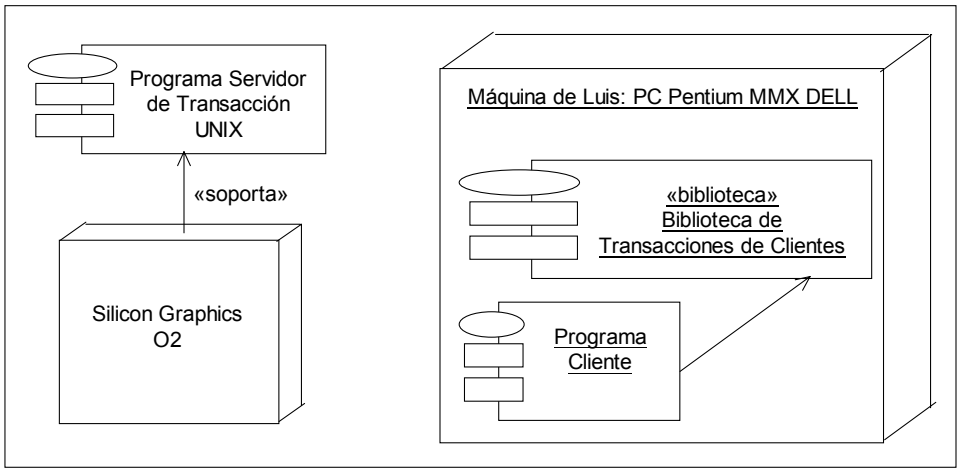


fig. 48 Un tipo nodo soportando un tipo componente y una instancia componente en tiempo de ejecución ejecutando en una instancia nodo

Los componentes pueden tener interfaces. Otros componentes pueden depender de las interfaces. [Booch 1999].

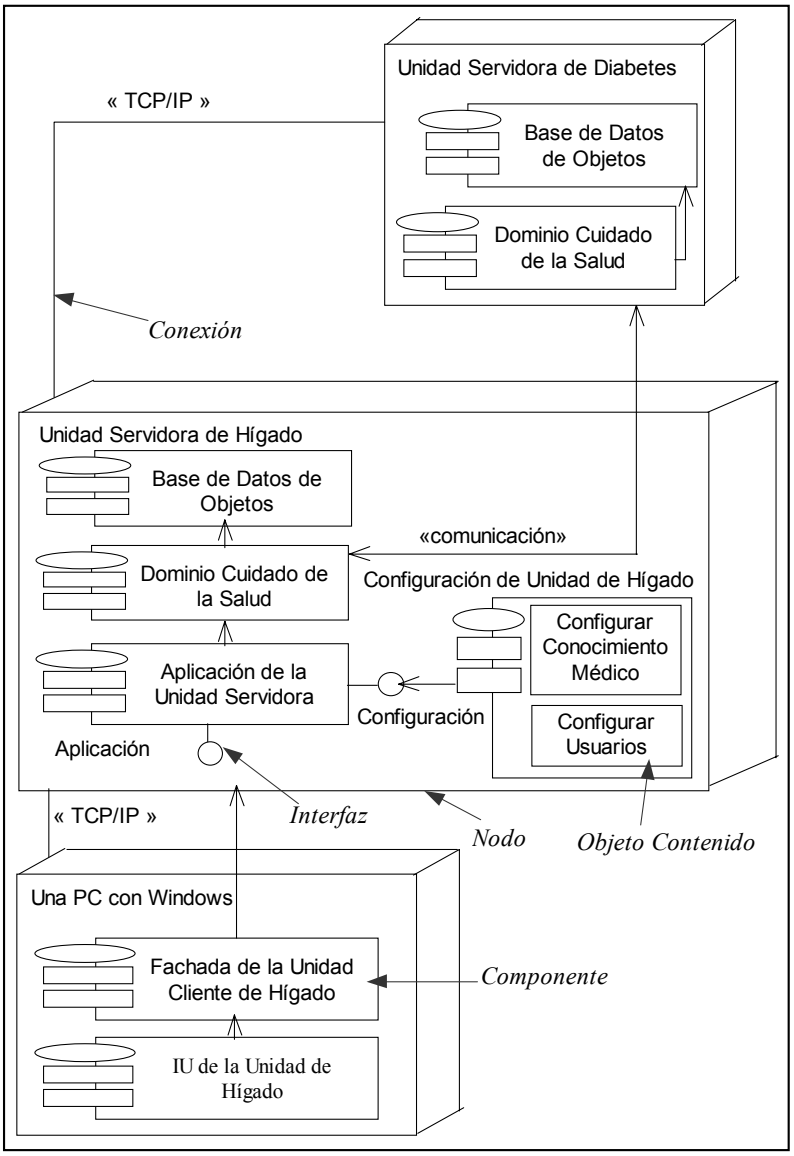


fig. 49 Diagrama de despliegue

6 Consideraciones en el uso de los diagramas

Los diagramas que dibujamos nos permiten visualizar un sistema desde diferentes perspectivas, por tal motivo representan proyecciones de diferentes aspectos del sistema. Modelan, como se mencionó anteriormente, los aspectos estático, dinámico y de arquitectura, necesarios para comprender e implementar el sistema.

Según sea la naturaleza del sistema, unos diagramas pueden ser más convenientes que otros. En sistemas intensivos en manejo de datos, los modelos que representan el aspecto estático serán de vital importancia y por tanto los dominantes. En sistemas intensivos en interfaces gráficas, los aspectos estáticos y dinámicos son fundamentales. En los sistemas de tiempo real las vistas de los procesos dinámicos tienden a ser más importantes que las otras vistas. Si estamos tratando con sistemas distribuidos, tales como las aplicaciones que encontramos en el Web, los modelos arquitectónicos y de implementación resultan ser vitales.

En otras palabras, aunque UML provee una gran cantidad de diagramas para el modelaje de los sistemas, tanto el área hacia la que se orienta la aplicación como su complejidad determinarán el tipo de diagrama que será necesario utilizar para modelar el sistema. A continuación se describe brevemente en qué casos es importante emplear los diagramas, mostrando algunas heurísticas que ayudan a tomar decisiones al respecto.

6.1 Diagrama de clases

El diagrama de clases ha sido el tipo de diagrama que ha estado presente en los diversos métodos que antecedieron al UML. Esto se debe a que representa el elemento central que guía el desarrollo de los otros diagramas al modelar las abstracciones presentes en el problema o la solución, así como las relaciones que existen entre ellas.

Uno de los problemas con el diagrama de clases radica en el abundante número de elementos que posee para realizar la descripción estática del sistema, que puede dar como resultado un diagrama sobrecargado e incomprensible. Por tanto es conveniente no utilizar toda la notación disponible. Se debe iniciar con los elementos simples (clases, asociaciones, atributos y relaciones de generalización), luego se podrán incluir elementos adicionales con mucho cuidado solamente si se hace necesario [Fowler 1997].

Debe analizarse la perspectiva desde la cual se está dibujando el modelo de acuerdo con la etapa de desarrollo. Si se está en la etapa de análisis, se deben dibujar modelos conceptuales; si se está trabajando con el software, concentrarse en el modelo de especificación o en el de implementación.

Entender la perspectiva es importante para dibujar y leer un diagrama de clases. Aunque las fronteras entre las diversas perspectivas no son absolutas, los objetivos con que cada una de ellas se realiza nos ayuda a centrarnos en sus aspectos esenciales. Mientras en la perspectiva conceptual se dibujan los diagramas para representar los conceptos en el dominio del problema, en la perspectiva de especificación nos interesan las interfaces del software, es decir cuáles serán las responsabilidades de los objetos que componen la arquitectura del sistema, que resultan

necesarias para obtener la funcionalidad que se demanda del sistema. Por último, las responsabilidades que han sido definidas para los objetos pueden ser implementadas de múltiples formas, dependiendo de los ambientes de implementación, las características de desempeño, etc. Especificar cómo se llevarán a cabo las responsabilidades es el centro de atención de la perspectiva de implementación.

6.2 Diagrama de casos de uso

Este tipo de diagrama representa una herramienta esencial para captar los requerimientos de los usuarios, planear y controlar un proyecto de forma iterativa. Identificar y establecer los casos de uso es una de las tareas iniciales que se debe emprender. La mayoría de los casos de uso se generarán en las etapas iniciales del proyecto, sin embargo se descubrirán nuevos casos de uso a medida que se avanza en el proceso de desarrollo. Cada caso de uso es un requerimiento potencial y hasta que sea *captado* el requerimiento se puede introducir como un caso de uso [Fowler 1997].

Los diagramas de casos de uso pueden ser de utilidad para los usuarios, diseñadores, desarrolladores y para quienes prueban el sistema. Resultan de interés para los usuarios debido a que los modelos de casos de uso especifican la funcionalidad del sistema y describen cómo el sistema puede ser usado. Ayudan a que los usuarios desempeñen un papel activo que permita plasmar sus deseos y necesidades en los modelos que se construyen. Debido a que los casos de usos se describen en un lenguaje y terminología entendible para clientes y usuarios, permiten que estos se comuniquen con los desarrolladores sin tener que penetrar en los detalles innecesarios y enfocarse en aquellos elementos de alto riesgo del sistema.

Los desarrolladores necesitan los casos de uso, por un lado, para comprender lo que el sistema debe realizar, por otro, para introducir en ellos las bases para la construcción de otro conjunto de modelos. Los casos de uso proveen la secuencia de eventos que los desarrolladores deberán implementar en un ambiente específico a fin de que el sistema cumpla con la funcionalidad deseada.

El equipo de prueba e integración utiliza los casos de uso para asegurarse que el sistema cumple con la funcionalidad especificada en esos casos de uso. Cualquiera que se encuentre involucrado en la funcionalidad del sistema tendrá interés en los modelos de casos de uso (los departamentos de mercadeo, de ventas, de soporte, etc.).

Los modelos de casos de uso no solamente captan los requerimientos de los nuevos sistemas; también pueden ser usados cuando se desarrollan nuevas versiones de un sistema existente. Cuando una nueva versión de un sistema se desarrolla, la nueva funcionalidad es agregada al modelo de casos de uso, insertando nuevos actores y casos de uso, o modificando la especificación de los casos de uso existentes. Cuando se agregue funcionalidad a un modelo de casos de uso existente se debe ser muy cuidadoso de no eliminar cualquier funcionalidad necesitada por el sistema.

Se debe tener presente que los objetivos fundamentales de los casos de uso consisten en [Eriksson 1998]:

- Decidir y describir los requerimientos funcionales del sistema, mediante un acuerdo entre los usuarios, quienes son los que tienen las necesidades, y los desarrolladores, que son quienes construyen el sistema.

- Proveer una descripción clara y consistente de lo que el sistema debe realizar, de tal forma que el modelo de casos de uso sea utilizado a través de todo el proceso de construcción para comunicar a los desarrolladores los requerimientos que proveerán las bases de los modelos subsecuentes.

- Proveer las bases para la ejecución de las pruebas del sistema. Por ejemplo, preguntar: ¿Cumple el sistema final con la funcionalidad inicialmente solicitada?

- Proveer la capacidad de rastrear los requerimientos funcionales en las clases y operaciones del sistema. Simplificar cambios y extensiones al sistema al permitir alterar el modelo de casos de uso y posteriormente rastrear los casos de uso afectados en el diseño e implementación del sistema.

6.3 Diagramas de interacción

Los diagramas de interacción corresponden a los diagramas de secuencia y diagramas de colaboración. Estos diagramas deberían utilizarse cuando se desea modelar el comportamiento y observar el flujo de control dentro de una operación, una clase, un caso de uso o del sistema como un todo [Booch 1999].

Los diagramas de actividad son otra forma de presentar interacciones pero enfocándose en el trabajo realizado por los objetos. Cuando los objetos interactúan, ejecutan un cierto trabajo en términos de actividades. En estos diagramas se describen esas actividades y su orden, así como el flujo de objetos. Los diagramas son más diagramas de estados, donde los estados son de actividad y las transiciones se disparan por la conclusión de actividades (y no por eventos externos).

Debido a que los diagramas de secuencia, colaboración y actividad presentan interacciones, frecuentemente se debe escoger cuál de los diagramas utilizar para documentar la interacción. La decisión depende de qué aspecto se desea representar. Cuando el énfasis se necesita centrar en el intercambio de mensajes que se suceden en el tiempo entre los objetos participantes, la opción es el diagrama de secuencia. Si el foco de atención se orienta hacia la relación estructural entre los objetos durante la interacción y lo que interesa es considerar cómo se pasan esos mensajes en el contexto de esa estructura, se debe optar por el diagrama de colaboración.

Los diagramas de secuencia son convenientes para especificaciones de tiempo real y para escenarios de casos de uso con situaciones complejas de interacción. Los diagramas de colaboración son mejores para comprender los efectos sobre un objeto dado y para el diseño de operaciones.

Los diagramas de actividad permiten: describir flujos debidos a procesamiento interno y no como respuesta a eventos externos, comportamientos con muchos hilos de control interactuantes y además especificar sincronización.

Fowler [Fowler 1997] refiriéndose a este tipo de diagramas expresa: "los diagramas de secuencia y colaboración son preferibles cuando el comportamiento es simple, ellos tienden a perder su claridad con comportamientos muy complejos (el comportamiento complejo se refiere a flujos de control interno de un objeto o hilo de control. El comportamiento conjunto puede ser complejo por la interacción que se da entre los objetos). Si se desea captar comportamientos complejos en un único diagrama es preferible utilizar los diagramas de actividad".

6.4 Diagrama de transición de estados

Los diagramas de transición de estados son útiles para mostrar los posibles estados por los que pasa un objeto en particular, como resultado de los eventos a los que puede responder. Son buenos para describir el comportamiento del objeto a través de varios casos de uso (servicios externos del sistema).

No son apropiados para describir el comportamiento que involucra un conjunto de objetos colaborando. Sin embargo se pueden combinar con otras técnicas, como los diagramas de interacción para observar la colaboración de diversos objetos en la implementación de un caso de uso o los diagramas de actividad para examinar la secuencia de acciones de diferentes objetos en un caso de uso.

No traten de usarse estos diagramas para describir el comportamiento de cada clase en el sistema, sino de aquellas clases que exhiben un comportamiento interesante, por ejemplo clases cuyas instancias necesitan recordar ciertos estados en los que se han encontrado, o también clases que desempeñan el papel de interfaces de usuario donde resulta conveniente describir los estados que ocurren ante los eventos generados por el usuario.

6.5 Diagrama de paquetes

Debido a que un diagrama de paquetes presenta generalmente grupos de clases y las dependencias que entre ellos se suscitan, es también una forma de diagrama de clases. Estos diagramas resultan ser una herramienta de utilidad para agrupar clases relacionadas en proyectos grandes. Se pueden emplear asimismo, para presentar diferentes vistas de la arquitectura del sistema.

Paquetes bien diseñados agrupan las clases que se encuentran semánticamente relacionadas y que tienden a cambiar de forma conjunta. Los paquetes deben por tanto estar débilmente acoplados (a los otros paquetes) y ser fuertemente cohesivos (internamente), permitiendo un acceso restringido y controlado al contenido del paquete [Booch 1999].

Se puede utilizar generalización con los paquetes, lo cual significa que el paquete específico debe ajustarse a la interfaz del paquete general. Una relación de generalización implica una dependencia del subtipo hacia el supertipo.

6.6 Diagramas de implementación

Se presentan en dos variedades: como diagramas de componentes y como diagramas de despliegue. Los diagramas de componentes muestran las dependencias entre componentes de software. Los componentes habitan en el mundo material de los bits y representan módulos físicos de código, tales como ejecutables, bibliotecas y fuentes. Los diagramas de componentes son esencialmente diagramas de clases que se enfocan en los componentes del sistema.

Los diagramas de despliegue muestran la configuración de elementos físicos de procesamiento (nodos) en tiempo de ejecución, así como los componentes, procesos y objetos asignados a los nodos. Los nodos, al igual que los componentes, modelan los aspectos físicos del sistema, representan elementos que existen en tiempo de ejecución como recursos computacionales, generalmente cuentan con algún tipo de memoria y a menudo con capacidad de procesamiento.

Se utilizan los nodos para modelar la topología del hardware sobre la cual el sistema se ejecuta. El nodo típicamente representará un procesador o un dispositivo en el cual los componentes serán desplegados [Booch 1999].

Los diagramas de componentes ayudan a modelar la vista estática de implementación de un sistema. Esta vista soporta principalmente la configuración de las partes del sistema, conformada por los componentes que pueden ser ensamblados de diferentes maneras, para producir un sistema en ejecución.

Con respecto de los diagramas de despliegue, es conveniente usarlos si se está desarrollando una pieza de software que interactúa con dispositivos que el sistema operativo propietario no controla o que se encuentran distribuidos a través de múltiples procesadores. Emplear este tipo de diagrama ayudará a discernir acerca del mapeo hardware-software del sistema. Se pueden ignorar en el caso de desarrollos de piezas de software que existen sobre una máquina y que interactúan con dispositivos estándar que se encuentran en la máquina controlada por un sistema operativo propietario.

7 Conclusiones y recomendaciones

UML es el resultado de un proceso de unificación de los principales métodos de análisis y diseño que aparecieron a finales de los '80s e inicios de los '90s. Esta multitud de métodos y herramientas[2], cada uno con su propia notación y terminología, tenían a muchos desarrolladores confundidos. La falta de una notación bien establecida sobre la cual estos métodos y herramientas estuvieran de acuerdo creó dificultades en la escogencia del método adecuado [Ericksson 1998].

Uno de los conceptos iniciales detrás de UML fue poner fin a la llamada "guerra de los métodos" dentro de la comunidad de orientación a objetos. Inicialmente, cuando en 1994 James Rumbaugh (líder del grupo que creó OMT) se unió a Grady Booch (creador del método enfocado esencialmente al diseño e implementación) en la compañía Rational, se pretendía desarrollar un método (el Método Unificado). Sin embargo, al unírseles Ivar Jacobson en 1995 (el hombre detrás de OOSE y el método Objectory), consideraron que el trabajo debería estar dirigido a desarrollar un lenguaje de modelaje unificado y no un método, por lo que renombraron el trabajo a UML (Lenguaje de modelaje unificado).

Aunque las principales partes de UML se basan en los métodos de Booch, OMT y OOSE, sus creadores incluyeron conceptos que provienen de otros. Por ejemplo, el trabajo de David Harel alrededor de los diagramas de estados o partes de la notación de Fusion para diagramar colaboraciones, se incorporaron en UML.

Los objetivos de UML, como fueron declarados por sus diseñadores, son:

- Modelar sistemas (no únicamente software) utilizando conceptos orientados a objetos.
- Establecer un acoplamiento explícito entre los aspectos conceptuales y los artefactos ejecutables de un sistema.
- Manejar los problemas de escalabilidad inherentes a los sistemas complejos y de misión crítica.
- Crear un lenguaje de modelaje utilizable tanto por los humanos como por las máquinas.

En noviembre de 1997 el UML fue escogido por el OMG (Object Management Group) como notación estándar para la construcción de modelos orientados a objetos.

Se ha dado un avance en la ingeniería del software al disponer de un estándar para el modelaje con lo que se establece una manera apropiada de comunicarse entre los desarrolladores. Sin embargo, si no se comprende la importancia de crear modelos en el desarrollo de sistemas, vano resultará todo el esfuerzo realizado.

La necesidad e importancia de modelar han sido evidentes en todas las disciplinas ingenieriles. Siempre que se planea desarrollar un proyecto, se realizan diagramas que muestran la estructura y comportamiento de los elementos que lo componen. Esos elementos pueden ser una casa o

[2] Para 1992 se inventariaron los métodos existentes en ese entonces: ¡se contaban 23 métodos con la etiqueta "objeto" en su nombre!.

edificio, una máquina, dispositivos eléctricos, sistemas de riego, cuencas hidrográficas, circuitos electrónicos, etc.

Los diagramas representan una especificación de cómo queremos que luzca el producto final. Estos diagramas son manejados por las personas involucradas en el proyecto, y pueden refinarse en diagramas más detallados necesarios para el trabajo de construcción.

Planes para estimación de costos y tiempo, distribución del trabajo y ubicación de recursos, entre otros, se llevan a cabo basados en la información contenida en los diagramas. Estos diagramas son una simplificación de la realidad y corresponden a los modelos que se construyen para tener un pleno entendimiento de los productos/artefactos que se planean desarrollar.

Producir modelos es un trabajo creativo. No existe una solución final, no existe respuesta de la cual se pueda decir que es la única correcta, al final del trabajo. Los diseñadores de modelos realizan un proceso iterativo, asegurando que los modelos reflejen los objetivos y requerimientos del proyecto que se pretende desarrollar.

Uno de los principales problemas con los desarrolladores actuales es que muchos proyectos inician la programación muy pronto y concentran demasiados esfuerzos en escribir código; básicamente, lo que se piensa se codifica. En parte, esto es el resultado de falta de entendimiento de los jefes acerca del proceso de desarrollo del software, condición que los lleva a un estado de ansiedad cuando el equipo de desarrolladores no está produciendo código. Por otro lado, los programadores se sienten más seguros cuando se encuentran programando (tarea con la que se han familiarizado) que cuando construyen modelos abstractos del sistema.

Si se desea obtener todos los beneficios que la orientación a objetos ofrece, es necesario cambiar este panorama. Tiene que haber un cambio en la forma en que pensamos con respecto de la construcción de sistemas. Debemos observar y tomar en cuenta los beneficios que las otras disciplinas han conseguido del empleo de modelos que les permiten validar la teoría y reducir al mínimo los costos y riesgos involucrados.

Además, si esperamos que nuestros productos sean competitivos tienen que cumplir con ciertos estándares, los cuales miden la calidad del producto, los elementos que lo componen (subsistemas, arquitectura, etc) [Kruchten 1998] así como los artefactos (requerimientos, diseño, código fuente, planes, pruebas, prototipos, versiones) que resultan del proceso de desarrollo y que sirven de apoyo al producto principal. Cuando parte de estos artefactos son pobres e insuficientes o ni siquiera existen, ¿qué elementos medirán estos estándares?, ¿dónde queda la calidad de lo que producimos?

Se debe tomar una conciencia generalizada para llevar la ingeniería del software al nivel de disciplina, donde realmente se siga un riguroso proceso de desarrollo que, utilizando notaciones y conceptos estándar, sea capaz de producir modelos precisos, completos y sin ambigüedades. Donde cada una de las fases del proceso entregue un conjunto de artefactos que nos permitan validar si lo que se construye corresponde con los requerimientos de los usuarios y a la vez medir la calidad de lo que producimos. Hacer a un lado la visión que se tiene de crear sistemas de software únicamente como forma de arte y colocar a la ingeniería del software en el lugar que le

corresponde. Solamente transformando nuestra mentalidad podrán cambiar las perspectivas de desarrollo de los sistemas.

Referencias

Bibliografía

[Booch 1999] Booch, G.; Rumbaugh, J.; Jacobson, I. The Unified Modeling Language User Guide. Addison-Wesley Longman, 1999.

[Eriksson 1998] Eriksson, H-E; Penker, M. UML Toolkit. John Wiley & Sons, 1998.

[Fowler 1997] Fowler, M. UML Distilled. Addison-Wesley, 1997.

[Joyanes 1998] Joyanes. L. Programación orientada a objetos. McGraw-Hill, 1998.

[Kruchten 1998] Kruchten, P. The Rational Unified Process.. Addison-Wesley Longman, 1999.

[Ruble 1998] Ruble, D. Análisis y diseño práctico de sistemas Cliente/Servidor con GUI. Prentice Hall, 1998.

[Rational Software 1997] UML Notation Guide. Rational Software Corporation, 1997.

[Trejos 1999] Trejos, I.; Luna, A. El modelo de objetos: Análisis y Diseño. Informe N° 26 del Club de Investigación Tecnológica. 1999.

Direcciones en Internet

En las direcciones que se listan abajo se puede encontrar una variedad de información relacionada con el UML y otros temas, como el análisis y el diseño orientados a objetos.

http://www.rational.com

http://www.omg.org/uml/

http://www.cetus-links.org/oo_uml.html

http://home.pacbell.net/ckobryn/uml.htm

http://www.sdmagazine.com/uml/

http://home.earthlink.net/~salhir/applyingtheuml.html

http://www.ratio.co.uk/white.html

http://www2.awl.com/cseng/titles/0-201-89542-0/techniques/standards.htm