

Tema 5: Técnicas de Especificación y Modelado

Introducción a UML

BLOQUE II: ESPECIFICACIÓN DE REQUISITOS
Y ANÁLISIS DE LOS SISTEMAS SOFTWARE

Ingeniería del Software
Grado en Ingeniería Informática
Curso 2024/2025

David Cáceres Gómez



Índice

1. Introducción	2
1.1. El Rol de los Modelos en la Ingeniería del Software	2
1.2. Técnicas Orientadas a Objeto: Lenguaje Unificado de Modelado	4
1.3. El Modelo Conceptual de UML	7
1.3.1. Bloques básicos de construcción	7
1.3.2. Reglas de combinación	18
1.3.3. Mecanismos comunes	19
1.3.4. Arquitectura	20
2. El Modelo de Comportamiento en UML	22
2.1. Diagrama de Casos de Uso	22
2.2. Diagrama de Actividades	30
2.3. Diagrama de Máquinas de Estado	36
2.4. Diagrama de Interacción	41
2.4.1. Diagrama de Secuencia	44
2.4.2. Diagrama de Colaboración	48
3. El Modelo Estructural en UML	50
3.1. Diagrama de Clases	50
3.2. Diagrama de Objetos	58
3.3. Diagrama de Paquetes	61
3.4. Diagrama de Componentes	64
3.5. Diagrama de Despliegue	67
4. Técnicas Estructuradas	70
4.1. Clasificación según el Enfoque de Representación	70
4.2. Clasificación según el Enfoque de Modelado	71
4.2.1. Diagramas de Flujo de Datos	73
Referencias	80

1. Introducción

1.1. El Rol de los Modelos en la Ingeniería del Software

“Los modelos son los planos del software”, una representación abstracta que permite visualizar y razonar sobre un sistema antes de proceder a su implementación en código. Al igual que los planos de un arquitecto, los modelos ofrecen una visión estructurada del software, ayudando a los desarrolladores y otros interesados a comprender cómo funcionará el sistema antes de construirlo físicamente. Esto reduce riesgos y mejora la planificación, ya que permite evaluar decisiones de diseño sin necesidad de escribir código.

Una de las principales ventajas de los modelos es que describen aspectos clave del sistema, como su **estructura**, que define los componentes y su interrelación; su **comportamiento**, que explica cómo estos componentes interactúan y reaccionan ante diferentes estímulos; y su **despliegue**, que detalla cómo el sistema se distribuirá y ejecutará en diferentes entornos.

Además, los modelos facilitan la **abstracción** del sistema completo, lo que permite a los desarrolladores y diseñadores manejar la complejidad de los proyectos sin tener detalles innecesarios. Al simplificar el sistema en conceptos clave, se mejora la comunicación entre los miembros del equipo, lo que a su vez reduce la ambigüedad y los malentendidos que suelen surgir en la fase de diseño y desarrollo.

Estos modelos son útiles en todas las etapas del desarrollo del software. Se usan antes de la construcción para analizar y prever problemas, durante la implementación para guiar el proceso de programación y toma de decisiones, y después de su construcción para facilitar la documentación, el mantenimiento y futuras ampliaciones del sistema.

Otra de sus grandes ventajas es que, al contrario de los documentos textuales que pueden ser interpretados de forma ambigua, los modelos mejoran la precisión y claridad en la representación del sistema. Cumplen con los criterios establecidos por Stachowiak [1], quien establece que los modelos deben:

- Representar un objeto o fenómeno real.
- Actuar como un “espejo” parcial de la realidad, capturando solo los aspectos más relevantes para el propósito en cuestión.
- Reemplazar al original para cumplir alguna función específica, como análisis, simulación o predicción del comportamiento del sistema.

Ingeniería de Sistemas Basada en Modelos

La Ingeniería de Sistemas Basada en Modelos (MBSE) es un enfoque formal que utiliza modelos como principal herramienta para el desarrollo de sistemas complejos. En lugar de basarse principalmente en documentos textuales para capturar y comunicar las especificaciones y detalles del sistema, la MBSE emplea modelos que representan diferentes aspectos del sistema a lo largo de todo su ciclo de vida. Esto incluye desde el análisis de los requisitos iniciales hasta la implementación y operación del sistema en funcionamiento.

Uno de los pilares de la MBSE es que los modelos se convierten en el principal medio de intercambio de información entre los diferentes equipos involucrados en el desarrollo del sistema, superando a los documentos. Esta estrategia permite mejorar la coherencia, precisión y comprensión del sistema al eliminar posibles ambigüedades que podrían surgir en la documentación tradicional. Los modelos actúan como una fuente de información, asegurando que todos los involucrados en el proyecto estén trabajando con una representación compartida y actualizada del sistema.

Ingeniería del Software Guiada por Modelos

La Ingeniería del Software Guiada por Modelos (MDE) es una metodología de desarrollo de software que se basa en la creación de modelos que representan el dominio específico de la aplicación. La MDE está enfocada en la abstracción del conocimiento y las actividades del dominio, es decir, en capturar los conceptos y procesos clave de una manera que sea comprensible y manipulable a través de modelos formales.

Una de las ventajas principales de la MDE es que aumenta la productividad al permitir una mayor automatización en la generación de software, reduciendo la cantidad de código manual que se debe escribir. También fomenta la reutilización, ya que los modelos pueden ser adaptados y extendidos para futuros proyectos, mejorando la eficiencia del desarrollo en el largo plazo. Además, la MDE simplifica el diseño del software al centrarse en una representación de más alto nivel, donde los detalles técnicos quedan encapsulados dentro del modelo.

Un aspecto clave de la MDE es la capacidad de utilizar los modelos no solo para diseñar el sistema, sino también para generar código automáticamente, en un enfoque conocido como desarrollo dirigido por modelos (*Model-Driven Development*, MDD). Esta capacidad permite que los modelos se traduzcan directamente en implementaciones, acortando el ciclo de desarrollo y asegurando que el código esté alineado con el diseño conceptual del sistema. Además, los modelos pueden ser utilizados para probar el sistema, en lo que se denomina pruebas basadas en modelos (*Model-Based Testing*, MBT), lo que facilita la detección de errores y la validación del comportamiento del sistema desde las primeras etapas del desarrollo.

1.2. Técnicas Orientadas a Objeto: Lenguaje Unificado de Modelado

A mediados de los años noventa, el desarrollo de software orientado a objetos estaba marcado por la existencia de múltiples métodos de análisis y diseño, cada uno con su propia notación y enfoque. Aunque compartían muchos de los mismos conceptos, las diferencias en la notación y en la metodología generaban confusión y fragmentación en la industria. Este fenómeno fue conocido como la “guerra de los métodos”, ya que cada propuesta competía por ser la dominante, lo que dificultaba la comunicación y el avance en el desarrollo de software.

En 1994, tres de los principales autores de estos métodos —Grady Booch, James Rumbaugh e Ivar Jacobson— decidieron unificar sus enfoques, lo que dio lugar a la creación del **Unified Modeling Language (UML)**. Este lenguaje unificado buscaba proporcionar una notación estándar para modelar sistemas orientados a objetos, abordando tanto el análisis como el diseño. La estandarización de UML fue promovida por el Object Management Group (OMG), lo que consolidó su posición como el estándar de facto en el modelado de software.

Las raíces técnicas de UML provienen de la combinación de los tres enfoques:

- **Object Modeling Technique (OMT)** de Rumbaugh: estaba centrada en el análisis de datos, especialmente en sistemas de información, y utilizaba diagramas de entidad-relación extendidos para representar las relaciones entre objetos.
- **Método-Booch**: era ideal para sistemas concurrentes y de tiempo real, y estaba relacionado con lenguajes de programación como Ada, lo que lo hacía apto para sistemas embebidos y de alta complejidad.
- **Object-Oriented Software Engineering (OOSE)** de Jacobson: introdujo el concepto de desarrollo basado en casos de uso, una técnica muy poderosa para capturar los requisitos del sistema y planificar su comportamiento, con especial aplicación en sistemas de telecomunicaciones y en la Ingeniería de Requisitos.

La creación de UML unificó estos enfoques, combinando sus fortalezas y añadiendo nuevos elementos para mejorar su capacidad de modelado. De esta manera, UML no solo estandarizó la notación, sino que también permitió unificar los procesos de análisis y diseño de sistemas orientados a objetos, ofreciendo una plataforma común y flexible para el desarrollo de software a gran escala.

Evolución Histórica de UML

- En 1994 Rumbaugh y Booch crean el Método Unificado.
- En 1995 se incorpora Jacobson y los tres autores publican un documento titulado Unified Method v0.8.
- El Método Unificado se reorienta hacia la definición de un lenguaje universal para el modelado de objetos, transformándose en UML (Unified Modeling Language for Object-Oriented Development).
- En 1996 se crea un consorcio de colaboradores para trabajar en la versión 1.0 de UML.
- En 1997 se produce la estandarización de UML 1.0 por la OMG.
- La siguiente versión oficial de UML es la versión 1.1.
- En julio de 1998 aparece una revisión interna de UML que recoge diversos cambios editoriales, pero no técnicos. Esta versión es la que se conoce como UML 1.2.
- Casi un año más tarde, en junio de 1999 aparece OMG UML 1.3 con algunos cambios significativos, especialmente en lo tocante a la semántica.
- En septiembre de 2001 aparece UML 1.4 y en enero de 2005 OMG UML 1.4.2 (OMG document: formal/05-04-01) es aceptado como un estándar ISO (ISO/IEC 19501).
- En julio de 2005 se libera UML 2.0 siendo la última versión 2.5.1 (2017).

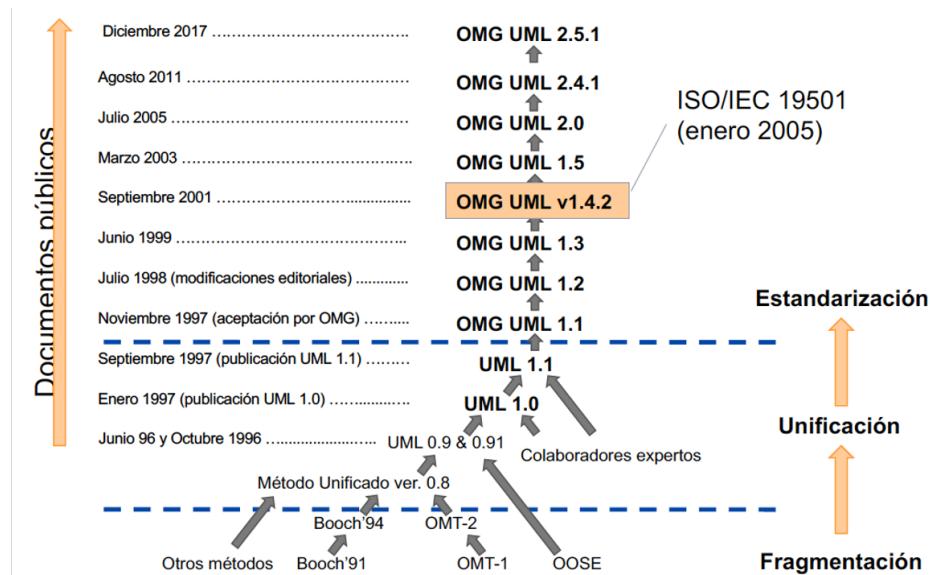


Figura 1: Evolución de UML

Ventajas de la Unificación

La unificación de los métodos orientados a objetos en UML ofreció diversas ventajas. Primero, **reunió los puntos fuertes de cada método** como OMT, Booch y OOSE, permitiendo combinar enfoques estructurados, concurrentes y basados en requisitos. Además, facilitó la **introducción de nuevas mejoras**, lo que sentó las bases para futuras innovaciones en el modelado de software. Esta unificación **proporcionó estabilidad al mercado**, eliminando la fragmentación anterior y ofreciendo un **lenguaje maduro** y ampliamente aceptado. También impulsó la creación de **potentes herramientas** de modelado y **redujo la confusión** entre los usuarios, promoviendo una notación estándar que mejoró la colaboración en proyectos.

Objetivos en el Diseño de UML

El diseño de UML tuvo varios objetivos fundamentales. El primero fue **modelar sistemas desde los requisitos hasta los artefactos ejecutables** mediante técnicas orientadas a objetos, abarcando todo el ciclo de vida del software. También se planteó **abordar la complejidad de sistemas grandes y críticos**, haciéndolo lo suficientemente flexible para proyectos de gran escala. Además, UML fue diseñado para ser **útil tanto para personas como para máquinas**, facilitando la comunicación y permitiendo la automatización. Finalmente, se buscó **equilibrar expresividad y simplicidad**, logrando un lenguaje potente sin ser excesivamente complicado, lo que favoreció su adopción masiva.

Definición de UML

UML es un lenguaje para **visualizar, especificar, construir y documentar** los artefactos (modelos) de un sistema que involucra una gran cantidad de software, desde una perspectiva orientada a objetos (OO).

Su principal función es proporcionar una notación para modelar diferentes aspectos del sistema, pero **no es un proceso**. UML sirve como una herramienta para representar de manera clara y consistente los componentes y relaciones dentro del sistema, facilitando la comprensión y comunicación entre los actores del desarrollo.

Se han definido muchos procesos que utilizan UML como base para el desarrollo de software. Uno de los procesos más conocidos es el Proceso Unificado de Rational (RUP), ideado por la empresa Rational.

Aunque inicialmente fue creado para proyectos de software, UML y RUP son **utilizables en sistemas más allá del software**, incluyendo áreas como ingeniería de sistemas y otras disciplinas donde es necesario modelar sistemas complejos.

1.3. El Modelo Conceptual de UML

Para comprender UML, es fundamental familiarizarse con sus elementos clave, que incluyen los **bloques básicos** compuestos por elementos, relaciones y diagramas. Además, es importante tener en cuenta las **reglas** de combinación, que dictan cómo se pueden ensamblar los bloques de manera coherente y significativa, y los **mecanismos comunes** que son aplicables en todo el lenguaje UML, garantizando la consistencia y la interoperabilidad entre diferentes modelos y diagramas. En la Figura 2 se puede observar un esquema de todos los elementos de UML.

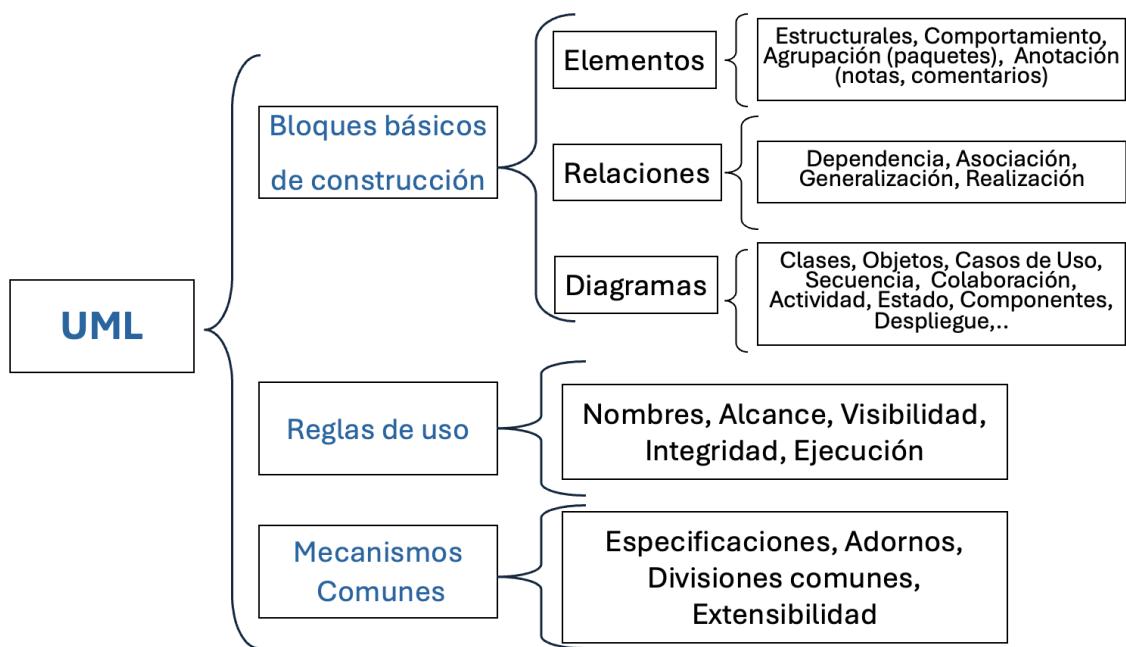


Figura 2: Elementos de UML [2]

1.3.1. Bloques básicos de construcción

UML se compone de tres bloques de construcción:

- **Elementos:** son bloques básicos de la programación orientada a objetos, abstracciones de primera clase dentro de un modelo. Tipos:
 - Elementos estructurales
 - Elementos de comportamiento
 - Elementos de agrupación
 - Elementos de anotación

- **Relaciones:** Conectan los diferentes elementos entre sí.
- **Diagramas:** Representación gráfica de un conjunto de elementos y sus relaciones entre sí.

Elementos estructurales

Los elementos estructurales, también llamados *clasificadores*, son los nombres de los modelos UML.

- En su mayoría son las partes estáticas de un modelo.
 - Representan conceptos o cosas materiales.
 - Elementos que representan cosas conceptuales o lógicas: clase, interfaz, colaboración, caso de uso, clase activa y componente.
 - Elementos que representan elementos físicos: artefactos y nodos.
- **Clase**

Una *clase* es una descripción de un conjunto de objetos que comparten los mismos atributos, operaciones, relaciones y semántica. Actúa como un clasificador que define tanto la estructura como el comportamiento de dichos objetos. Además, una clase puede implementar una o más interfaces.

Gráficamente, una clase se representa como un rectángulo que generalmente contiene su nombre, atributos y operaciones, como se muestra en la Figura 3.

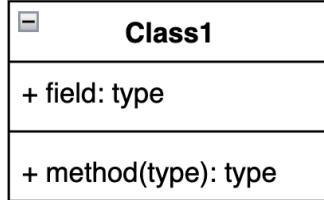


Figura 3: Elemento estructural *clase* [3]

- **Caso de uso**

Un *caso de uso* es una descripción de un conjunto de secuencias de acciones que ejecuta un sistema y que produce un resultado observable de interés para un actor particular. Un caso de uso se utiliza para estructurar los aspectos de comportamiento en un modelo. Un caso de uso es realizado por una colaboración. Gráficamente, un

caso de uso se representa como una elipse de borde continuo, y normalmente solo su nombre, como se muestra en la Figura 4.

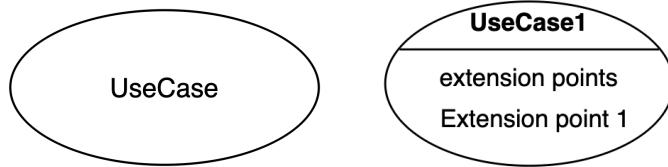


Figura 4: Elemento estructural *caso de uso* [3]

■ Interfaz

Una *interfaz* es un conjunto de operaciones que especifican un servicio de una clase o componente. Por tanto, una interfaz describe el comportamiento visible externamente de dicho elemento. Puede representar el comportamiento completo de una clase o componente, o solo una parte de este. Una interfaz define un conjunto de especificaciones de operaciones, pero nunca incluye la implementación de dichas operaciones.

La declaración de una interfaz es similar a la de una clase, pero utiliza la palabra clave «interface» antes del nombre. Los atributos no son relevantes en una interfaz, salvo en algunas ocasiones, para definir constantes. Por lo general, una interfaz no se encuentra aislada. Cuando una clase expone una interfaz al mundo exterior, esta se representa mediante un pequeño círculo conectado al rectángulo que simboliza la clase mediante una línea. Si una clase requiere una interfaz que será proporcionada por otra, dicha interfaz se representa como un semicírculo unido al rectángulo de la primera clase a través de una línea, como se muestra en la Figura 5.

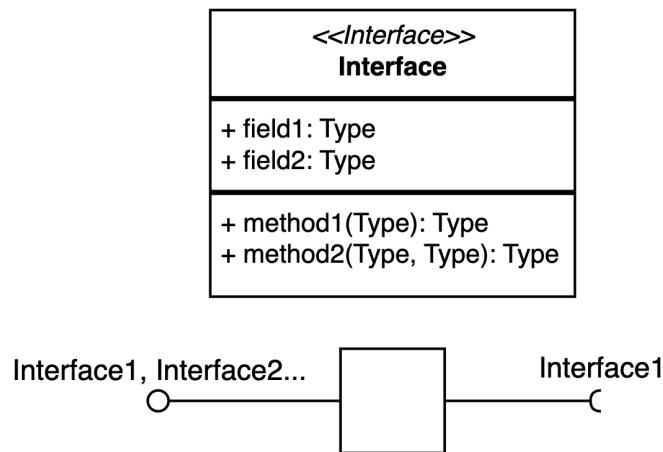


Figura 5: Elemento estructural *interfaz* [3]

■ Componente

Un *componente* es una unidad modular del diseño de un sistema que oculta su implementación detrás de un conjunto de interfaces externas definidas, las cuales se exponen a través de puertos. Los componentes que comparten las mismas interfaces pueden ser sustituidos en un sistema siempre que mantengan el mismo comportamiento lógico. Su interior permanece oculto y solo es accesible mediante las interfaces que proporciona. Además, un componente puede indicar dependencias a través de interfaces requeridas.

La implementación de un componente puede describirse mediante la conexión de partes y conectores, donde las partes pueden incluir componentes más pequeños. Gráficamente, un componente se representa como una clase con un ícono especial en la esquina superior derecha, como se muestra en la Figura 6.

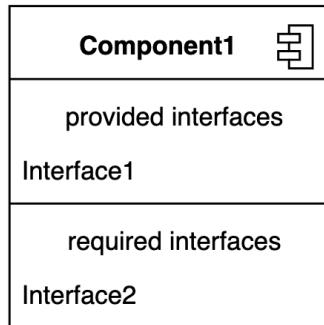


Figura 6: Elemento estructural *componente* [3]

■ Artefacto

Un *artefacto* es una parte física y reemplazable de un sistema que contiene información física (“bits”). En un sistema hay diferentes tipos de artefactos de despliegue, como archivos de código fuente, ejecutables y scripts. Un artefacto representa típicamente el empaquetamiento físico de código fuente o información en tiempo de ejecución. Gráficamente, un artefacto se representa como un rectángulo con la palabra clave «artifact» sobre el nombre, como se muestra en la Figura 7.

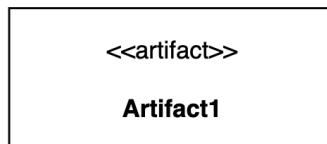


Figura 7: Elemento estructural *artefacto* [3]

■ Nodo

Un *nodo* es un elemento físico que existe en tiempo de ejecución y representa un recurso computacional, generalmente con memoria y capacidad de procesamiento. Los nodos pueden alojar un conjunto de artefactos, los cuales pueden migrar de un nodo a otro. También pueden estar compuestos por otros nodos internamente.

Los nodos representan dispositivos de hardware o entornos de software de ejecución, sirviendo como el recurso computacional sobre el cual se despliegan los artefactos para su funcionamiento. Gráficamente, un nodo se representa como un cubo, usualmente con solo su nombre, como se muestra en la Figura 8.

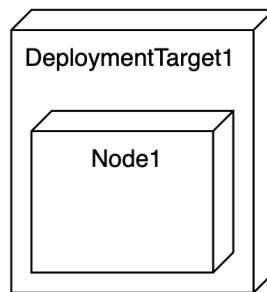


Figura 8: Elemento estructural *nodo* [3]

Elementos de comportamiento

Los elementos de comportamiento son las partes dinámicas en los modelos UML. Actúan como los “verbos” del modelo y representan el comportamiento a lo largo del tiempo y el espacio. Semánticamente, estos elementos están vinculados a elementos estructurales, como clases, colaboraciones y objetos.

■ Mensaje

Una *interacción* es un comportamiento compuesto por mensajes intercambiados entre objetos en un contexto específico, con el fin de lograr un objetivo determinado. Tanto el comportamiento de un conjunto de objetos como el de una operación individual pueden especificarse mediante una interacción, que incluye elementos como mensajes, acciones y enlaces (conexiones entre objetos).

Gráficamente, un *mensaje* se representa como una línea con dirección, generalmente acompañada del nombre de la operación asociada (Figura 9). Los mensajes actúan como el medio de comunicación para intercambiar información o expresar interacciones entre elementos, y pueden variar en tipo (respuesta, destrucción) y propiedades (asíncrono o no, solo lectura), lo cual afecta ligeramente su notación.

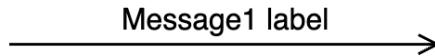


Figura 9: Elemento de comportamiento *mensaje* [3]

■ Estado

Una *máquina de estados* modela el comportamiento de un objeto o interacción a través de las secuencias de estados que atraviesa durante su vida, en respuesta a eventos, y sus reacciones ante estos. Puede describir el comportamiento de una clase o la colaboración entre varias clases.

La máquina de estados se compone de elementos como estados, transiciones (el paso de un estado a otro), eventos (que activan una transición) y actividades (acciones en respuesta a una transición). Gráficamente, un estado se representa como un rectángulo con esquinas redondeadas que incluye su nombre y, si los tiene, sus subestados, como se muestra en la Figura 10.



Figura 10: Elemento de comportamiento *estado* [3]

■ Acción

Una *actividad* es un comportamiento que especifica la secuencia de pasos en un proceso computacional. A diferencia de las interacciones, que se enfocan en los objetos que interactúan, o de las máquinas de estados, que se centran en el ciclo de vida de un objeto, una actividad se enfoca en los flujos entre pasos, sin considerar qué objeto ejecuta cada uno.

Cada paso dentro de una actividad se llama *acción*. En UML, las acciones son unidades fundamentales de comportamiento, generalmente con un conjunto de entradas y salidas que modifican el estado del sistema. Gráficamente, una acción se representa como un rectángulo con esquinas redondeadas, identificado por un nombre que indica su propósito, como se muestra en la Figura 11.

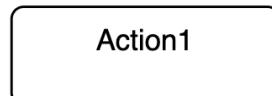


Figura 11: Elemento de comportamiento *acción* [3]

Elementos de agrupación

Un *paquete* es un mecanismo general para organizar y estructurar el diseño en UML, en contraste con las clases, que estructuran la implementación. Puede incluir elementos estructurales, de comportamiento e incluso otros paquetes. A diferencia de los componentes, que existen en tiempo de ejecución, un paquete es conceptual y solo existe durante el desarrollo.

Los paquetes actúan como “espacios de nombres” que delimitan el alcance de sus elementos. Además, se pueden establecer relaciones entre ellos (import, merge...). Gráficamente, un paquete se representa como una carpeta, que suele mostrar su nombre y, a veces, su contenido, como se ilustra en la Figura 12.

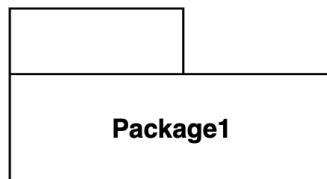


Figura 12: Elemento de agrupación *paquete* [3]

Elementos de anotación

Los elementos de anotación son componentes explicativos en los modelos UML, usados para describir, aclarar o hacer observaciones sobre cualquier elemento. El principal tipo de elemento de anotación es la *nota*, un símbolo que muestra restricciones y *comentarios* relacionados con un elemento o grupo de elementos. Estos elementos sirven únicamente para aclarar y no afectan la semántica del modelo.

Gráficamente, una nota se representa como un rectángulo con una esquina doblada y contiene texto o gráficos explicativos, como se muestra en la Figura 13.

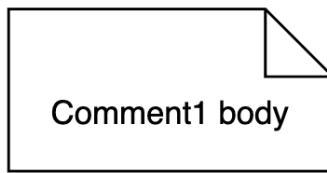


Figura 13: Elemento de anotación *nota* o *comentario* [3]

Relaciones

Las relaciones en UML permiten modelar los enlaces entre diferentes elementos estructurales del sistema, proporcionando una manera de representar sus interacciones y asociaciones. Estas relaciones también aportan información adicional, que resulta clave para comprender cómo se vinculan y operan los distintos elementos del modelo.

Una de las principales características que se puede especificar en una relación es la **multiplicidad**. Esta indica la cantidad de instancias de una clase que pueden estar asociadas con instancias de otra. La multiplicidad se expresa mediante notaciones específicas, tales como:

- ‘1’: indica que puede existir una única instancia.
- ‘0..1’: indica que puede haber entre cero y una instancia.
- ‘0..*’: indica que puede haber entre cero y muchas instancias.
- ‘1..*’: indica que debe haber al menos una instancia, pero puede haber muchas.
- ‘*’: indica un número indefinido de instancias.

Otra característica importante son los **nombres de roles**, que identifican los extremos de una asociación. Esto facilita entender el papel específico que cada elemento juega dentro de la relación.

En UML, se manejan cuatro clases principales de relaciones: **dependencia, asociación, generalización y realización**. Cada una de estas representa un tipo de conexión particular entre elementos, permitiendo definir de manera precisa cómo interactúan y dependen entre sí dentro del sistema modelado.

▪ Relación de Dependencia

Una *dependencia* es una relación semántica entre dos elementos, en la cual un cambio a un elemento (independiente) puede afectar a la semántica del otro elemento (dependiente). Gráficamente, una dependencia se representa como una línea discontinua, posiblemente dirigida, que incluye a veces una etiqueta, como se muestra en la Figura 14.



Figura 14: Relación de *dependencia*

■ Relación de Asociación

Una *asociación* es una relación estructural entre clases que describe un conjunto de enlaces, los cuales son conexiones entre objetos que son instancias de clases.

Gráficamente, una asociación se representa como una línea continua (Figura 15), posiblemente dirigida, que a veces incluye una etiqueta, y a menudo incluye otros adornos, como la multiplicidad y los nombres de rol.

Figura 15: Relación de *asociación*

■ Relación de Agregación

La *agregación* es un tipo especial de asociación en UML que representa una relación estructural entre un todo y sus partes. Es decir, modela una conexión donde una clase (el “todo”) está compuesta por una o más instancias de otra clase (sus “partes”). En una agregación, las partes pueden existir independientemente del todo; por ejemplo, un objeto “Equipo” podría estar compuesto por varios objetos “Jugador”, pero los jugadores podrían existir fuera del equipo.

Gráficamente, la agregación se representa como una línea de asociación con un rombo vacío en el extremo que indica el “todo” (la clase contenedora), como se muestra en la Figura 16.



Figura 16: Relación de *agregación*

■ Relación de Composición

La *composición* es un caso más fuerte de agregación, donde también se representa una relación “todo-parte”, pero con una dependencia de vida más estricta entre las partes y el todo. En una composición, las partes no pueden existir independientemente del todo; si el objeto contenedor se destruye, sus partes también dejan de existir. Por ejemplo, en una relación de composición entre “Casa” y “Habitación”, la “Habitación” no existiría si no existe la “Casa” que la contiene.

Gráficamente, la composición se representa con una línea de asociación que tiene un rombo sólido en el extremo del “todo”, como se muestra en la Figura 17.



Figura 17: Relación de *composición*

■ Relación de Generalización

Una *generalización* es una relación de especialización/generalización en la cual el elemento especializado (el hijo) se basa en la especificación del elemento generalizado (el padre). El hijo comparte la estructura y el comportamiento del padre.

Gráficamente, una relación de generalización se representa como una línea continua como una punta de flecha vacía apuntando al padre, como se muestra en la Figura 18.

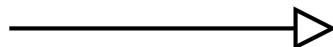


Figura 18: Relación de *generalización*

■ Relación de Realización

Una *realización* es una relación semántica entre clasificadores, en donde un clasificador especifica un contrato que otro clasificador garantiza que cumplirá. Se pueden encontrar relaciones de realización en dos contextos: entre interfaces y las clases o componentes que las realizan, y entre los casos de uso y las colaboraciones que los ejecutan.

Gráficamente, una relación de realización se representa como una mezcla entre una generalización y una relación de dependencia, como se muestra en la Figura 19.



Figura 19: Relación de *realización*

La Figura 20 muestra dos ejemplos de diagramas UML, con sus elementos y relaciones.

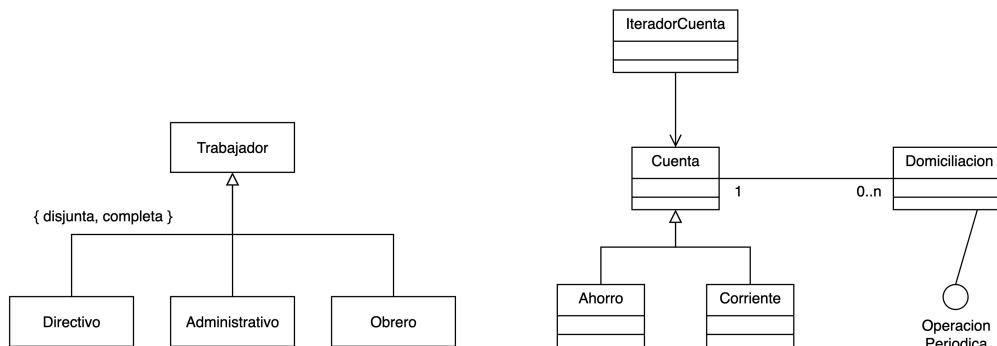


Figura 20: Ejemplos de elementos y relaciones

Diagramas

Un **diagrama** es una representación gráfica que muestra un conjunto de elementos conectados, permitiendo visualizar sus relaciones y la estructura general del sistema. Estos diagramas forman grafos conectados en los que los vértices representan los elementos del sistema y los arcos muestran las relaciones entre ellos. Mediante esta estructura, los diagramas ofrecen una manera de visualizar el sistema desde diferentes perspectivas, lo que facilita la comprensión de su arquitectura y su funcionamiento.

En la práctica, un mismo elemento puede aparecer en varios diagramas si es relevante en diferentes contextos, en algunos si es necesario en perspectivas específicas, o en ninguno si no es necesario en la visualización. Los diagramas se agrupan por:

- **Diagramas Estructurales:** son una categoría de diagramas que visualizan, especifican, construyen y documentan los aspectos estáticos de un sistema. Estos representan la organización y disposición de los elementos que conforman la estructura del sistema, como clases, componentes y paquetes, proporcionando una vista detallada de la arquitectura subyacente.
- **Diagramas de Comportamiento:** están enfocados en los aspectos dinámicos del sistema. Su objetivo es visualizar, especificar, construir y documentar el comportamiento del sistema en tiempo real, incluyendo la interacción entre los elementos y la secuencia de acciones y eventos. Así, cada tipo de diagrama cumple un rol específico en la representación y el análisis de diferentes facetas del sistema modelado.

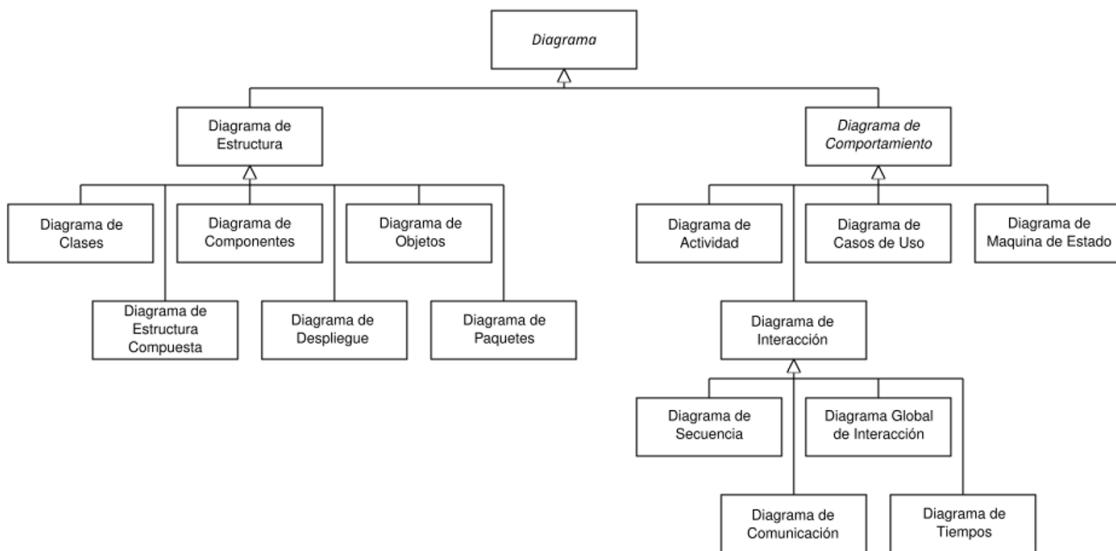


Figura 21: Tipos de diagramas

1.3.2. Reglas de combinación

Los bloques de construcción de UML no se pueden combinar de manera específica. Un **modelo bien formado** es aquel que es semánticamente autoconsistente y está en armonía con todos sus modelos relacionados.

UML tiene un número de reglas semánticas que especifican a qué debe parecerse un modelo bien formado:

- **Nombres:** Especifican cómo llamar a los elementos, relaciones y diagramas.
- **Alcance:** Define el contexto que da un significado específico a un nombre.
- **Visibilidad:** Indica cómo se pueden ver y utilizar esos nombres por otros.
- **Integridad:** Asegura que los elementos se relacionen de manera apropiada y consistente entre sí.
- **Ejecución:** Establece lo que significa ejecutar o simular un modelo.

Según Booch [3], es habitual que el equipo de desarrollo no sólo construya modelos bien formados sino que los modelos sean:

- **Abreviados:** algunos elementos se ocultan para simplificar la vista.
- **Incompletos:** pueden estar ausentes ciertos elementos.
- **Inconsistentes:** no se garantiza la integridad del modelo.

Los autores afirman que estos modelos que no llegan a ser *bien formados* son inevitables según aparecen detalles del sistema y se avanza en el desarrollo. **Las reglas de UML favorecen (pero no obligan)** a considerar las cuestiones más importantes de análisis, diseño e implementación que permiten obtener sistemas bien formados con el paso del tiempo.

1.3.3. Mecanismos comunes

UML incorpora ciertos mecanismos comunes que se pueden aplicar en distintos modelos para mejorar su interpretación y claridad. Estos mecanismos son los siguientes:

1. **Especificaciones:** Proporcionan una base semántica que incluye a todos los modelos de un sistema, asegurando la coherencia entre los elementos. Se utilizan para detallar las características del sistema; cada elemento gráfico tiene una explicación textual que describe su sintaxis y semántica.
2. **Adornos:** Son notaciones gráficas que añaden detalles adicionales para aclarar o complementar la información de los elementos UML. Por ejemplo, el uso de cursiva para el nombre de una clase indica que es abstracta, mientras que el símbolo '+' indica que un método o atributo es público.
3. **Divisiones comunes:** Permiten modelar tanto desde una perceptiva general (abstracción) como particular (concreto). Esta opción está disponible en la mayoría de los bloques de construcción. Ejemplos incluyen:
 - Clase / Objeto
 - Casos de Uso / Instancias Casos de Uso
 - Componentes / Instancias de Componentes
4. **Extensibilidad:** Aunque UML proporciona un lenguaje estándar para el modelado de sistemas, admite adaptaciones para cubrir necesidades específicas de cada proyecto. Los mecanismos de extensibilidad incluyen:
 - **Estereotipos:** Crean nuevos tipos de bloques de construcción derivados de los existentes.
 - **Valores etiquetados:** Añaden nueva información a las propiedades de los elementos.
 - **Restricciones:** Extienden la semántica de un bloque de construcción para añadir o modificar reglas.

1.3.4. Arquitectura

La visualización, especificación, construcción y documentación de sistemas complejos requieren múltiples perspectivas. Las diferentes personas involucradas en el desarrollo tienen intereses variados y observan el sistema desde distintas perspectivas a lo largo de su ciclo de vida. En UML, la arquitectura de un sistema se describe óptimamente a través de cinco vistas interrelacionadas, cada una enfocada en un aspecto particular del sistema.

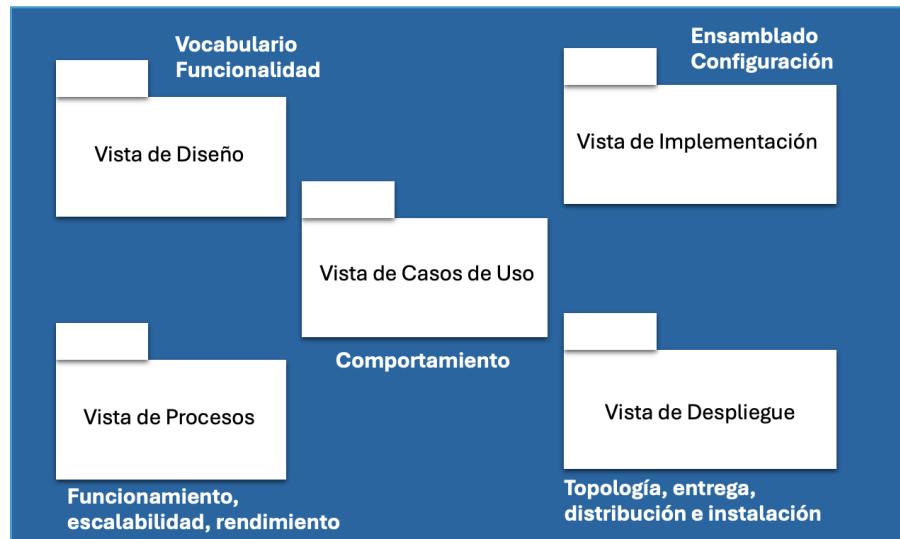


Figura 22: Vistas interrelacionadas

Vista de Casos de Uso

Esta vista describe el comportamiento del sistema desde la perspectiva de los usuarios finales, analistas y responsables de pruebas. Utiliza casos de uso para representar aspectos estáticos y diagramas de interacción, de estados y de actividades para los aspectos dinámicos.

Vista de Diseño

Incluye las clases, interfaces y colaboraciones que definen el vocabulario del problema y su solución, enfocándose principalmente en los requisitos funcionales del sistema. Para los aspectos estáticos, utiliza diagramas de clases y de objetos; para los aspectos dinámicos, se apoya en diagramas de interacción, de estados y de actividades.

Vista de Procesos

Esta vista abarca los hilos y procesos que gestionan la sincronización y la concurrencia, atendiendo al crecimiento, rendimiento y operación del sistema. Utiliza los mismos diagramas que la vista de diseño, pero se centra en clases activas que representan hilos y procesos.

Vista de Implementación

Describe los componentes y archivos necesarios para ensamblar y ejecutar el sistema físico, con un enfoque en la gestión de versiones de componentes independientes que pueden combinarse de distintas maneras. Utiliza diagramas de paquetes para los aspectos estáticos, y diagramas de interacción, de estados y de actividades para los aspectos dinámicos.

Vista de Despliegue

Muestra los nodos que componen la topología de hardware del sistema, centrándose en la distribución, entrega e instalación de los elementos del sistema físico. Para los aspectos estáticos, utiliza diagramas de despliegue, y para los aspectos dinámicos, diagramas de interacción, de estados y de actividades.

Cada una de estas vistas puede existir de forma independiente y también interactuar entre sí

2. El Modelo de Comportamiento en UML

2.1. Diagrama de Casos de Uso

Un **caso de uso** es una descripción detallada de cómo un sistema lleva a cabo una serie de acciones para generar un resultado valioso para un **actor**. Los casos de uso capturan el comportamiento esperado del sistema de manera que sea comprensible para desarrolladores, expertos en el dominio y usuarios finales, sin especificar detalles técnicos de implementación.

Ayudan a validar la arquitectura y a verificar el sistema a medida que evoluciona, asegurando que los requisitos del usuario se cumplan en cada etapa. Los casos de uso deben enfocarse en los comportamientos esenciales, evitando descripciones demasiado amplias o específicas.

Los casos de uso se pueden utilizar como base para establecer casos de prueba durante el desarrollo del sistema, pueden ser:

- **Casos de uso aplicados a subsistemas:** útiles para pruebas de regresión, permitiendo verificar que las funcionalidades existentes no se vean afectadas por nuevas adiciones o cambios.
- **Casos de uso aplicados al sistema completo:** ideales para pruebas de integración y pruebas del sistema, ya que permiten validar que todas las partes del sistema trabajan correctamente en conjunto.

Construcción del Modelo de Casos de Uso

La creación de un modelo de casos de uso sigue una serie de pasos que permiten estructurar y definir el sistema y sus interacciones de manera efectiva. Estos pasos son:

1. **Establecer el límite del sistema:** Identificar el alcance del sistema y los límites que lo separan de otros sistemas o del entorno externo.
2. **Definir los actores:** Determinar todos los usuarios o sistemas externos que interactuarán con el sistema.
3. **Identificar los casos de uso:**
 - Especificar el caso de uso: Describir detalladamente cada caso de uso para reflejar el comportamiento esperado del sistema.

- Incluir flujos alternativos clave: Identificar posibles variaciones en el flujo de trabajo principal, como excepciones o alternativas.
4. **Repetir los pasos** anteriores hasta que el sistema, los actores y los casos de uso sean estables y reflejen todos los requerimientos identificados.

Componentes del modelo

- **Límite del Sistema:** Define el alcance del sistema que se está modelando, delimitado visualmente mediante un rectángulo.
- **Actores:** Son los roles de personas o sistemas externos que interactúan con el sistema. Los actores tienen las siguientes características:
 - Una misma persona puede adoptar distintos roles, actuando como diferentes actores según el contexto.
 - Los **actores principales** son aquellos que activan los casos de uso, mientras que los **actores secundarios** solo interactúan sin activarlos.
 - El nombre del actor debe reflejar claramente su papel en la interacción con el sistema.
 - Se representan comúnmente como un “muñeco de palo” para mayor claridad en el diagrama.
 - Los actores pueden ser especializados mediante relaciones de generalización, permitiendo agrupar roles similares.
- **Casos de Uso:** Los casos de uso representan las acciones o funcionalidades que los actores pueden realizar con el sistema. Estos se representan visualmente como elipses.
- **Relaciones:** Las relaciones conectan a los actores con los casos de uso, mostrando el tipo de comunicación o interacción que existe entre ellos.

Un **diagrama de casos de uso** es una representación gráfica que ilustra un conjunto de casos de uso, actores, y las relaciones entre ellos.

Al igual que otros tipos de diagramas, puede incluir notas y restricciones para aclarar detalles específicos.

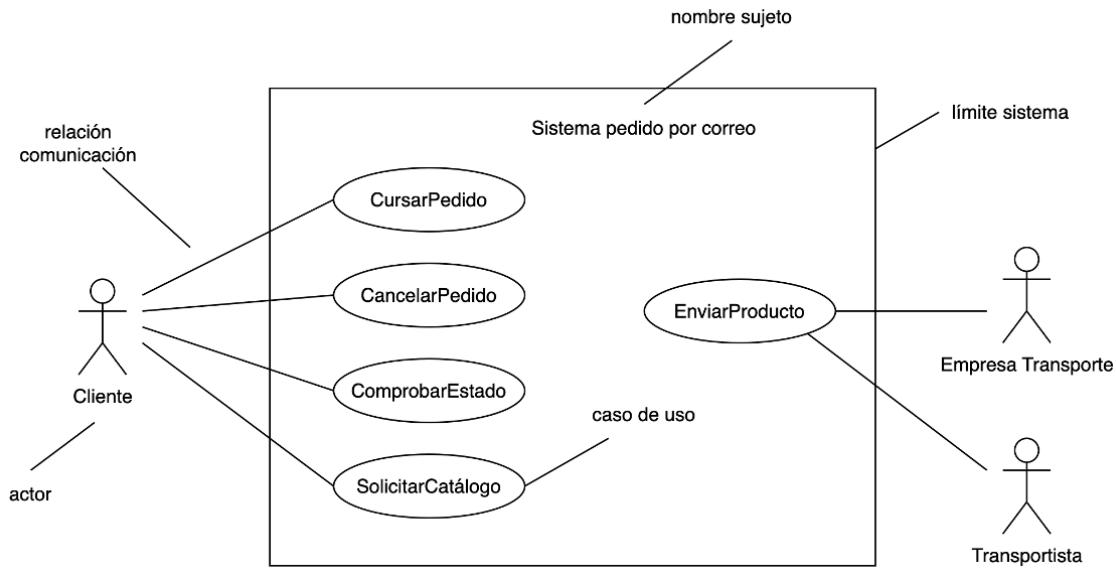


Figura 23: Ejemplo Diagrama Casos de Uso [4]

Especificación de los casos de uso

La especificación de un caso de uso describe su comportamiento a través de un flujo de eventos claro y comprensible para cualquier persona, incluso aquellas sin conocimientos profundos del sistema.

En este flujo de eventos se debe incluir los siguientes elementos:

- **Inicio y fin del caso de uso:** Indica cuándo comienza y finaliza la interacción.
- **Interacción con actores:** Especifica cómo interactúan los actores con el sistema.
- **Objetos intercambiados:** Detalla los datos o elementos que se intercambian entre el sistema y los actores.

Además, el flujo de eventos debe contener tanto el **flujo principal** (el curso normal de acciones) como los **flujos alternativos** que representan variaciones en el comportamiento.

Los casos de uso se identifican analizando las secuencias de interacción con los actores desde la perspectiva del usuario final.

Aunque no existe un formato estandarizado para documentar los casos de uso, es común describir la especificación en formato tabular con los siguientes campos:

- **Nombre** del caso de uso
- **ID** del caso de uso
- **Objetivo:** una breve descripción del propósito del caso de uso.
- **Contexto o precondiciones:** las condiciones previas necesarias.
- **Actores implicados:** especificando el actor principal y, si es necesario, los actores secundarios.
- **Flujo principal:** el escenario principal de los eventos.
- **Flujos alternativos:** extensiones del flujo principal.
- **Postcondiciones:** los resultados que se esperan una vez concluido el caso de uso.

En la Tabla 1 se puede observar un ejemplo de la especificación del caso de uso: Cancelar pedido.

Caso de uso	Cancelar pedido
Identificador	CU3
Objetivo	El cliente desea cancelar un pedido realizado anteriormente que aún no ha recibido
Contexto	El pedido debe existir en el sistema y no haberse tramitado aún
Actor principal	Cliente
Flujo principal	<ol style="list-style-type: none">1. El sistema solicita al cliente el código del pedido2. El cliente introduce el código del pedido a cancelar3. El sistema localiza los datos del pedido4. El cliente confirma que quiere eliminar el pedido5. El sistema elimina los artículos del pedido6. El sistema devuelve el dinero del pedido
Flujos alternativos	<ol style="list-style-type: none">3a. El pedido no existe. Se termina el caso de uso.3b. El pedido se encuentra en estado enviado. Se termina el caso de uso sin cancelar el pedido.

Tabla 1: Ejemplo caso de uso: Cancelar pedido

Para estructurar un caso de uso, es importante analizar cada paso y diferenciar claramente el flujo principal de los flujos alternativos.

- **Flujo principal:** Representa el escenario ideal en el que el caso de uso se ejecuta sin errores, desvíos o interrupciones. Puede tener desviaciones:

- Simples: Son ramificaciones que ocurren dentro del flujo principal.
 - Complejas: Escenarios alternativos que presentan rutas de acción diferentes.
- **Flujos alternativos:** Se enfocan en la gestión de errores y excepciones, y generalmente no regresan al flujo principal. Para una especificación clara, es útil:
- Seleccionar solo los flujos alternativos más relevantes.
 - Agrupar los flujos similares para evitar redundancias y simplificar el modelo.

Tipos de relaciones en UML:

Los diagramas de casos de uso en UML representan distintas relaciones entre actores y casos de uso que ayudan a organizar y simplificar el modelo del sistema. A continuación, se detallan los tipos de relaciones principales:

1. Comunicación

- Descripción: Relación básica entre los actores y el sistema, que define las entradas y salidas de información en el diagrama.
- Uso: Especifica cómo los actores se comunican con el sistema.

2. Herencia

- **Entre Actores:** Permite abstraer similitudes en la interacción de diferentes actores con el sistema.
- **Entre Casos de Uso:** Los casos de uso hijos pueden heredar, modificar o añadir características al caso de uso padre.
 - En UML 2, la herencia entre casos de uso afecta solo a relaciones, precondiciones, postcondiciones y flujos, ya que los casos de uso no tienen atributos ni operaciones.
 - **Recomendación:** Usar herencia únicamente cuando simplifique el modelo, ya que un uso excesivo puede dificultar su comprensión y mantenimiento.

3. Inclusión

- **Descripción:** Un caso de uso base incluye el comportamiento de otro caso de uso en un punto específico, fomentando la reutilización.
- **Evita duplicación:** Facilita la reutilización de flujos de eventos comunes mediante un caso de uso independiente.
- **Representación:** La relación de inclusión se denota con la etiqueta «include».
- El caso de uso incluido siempre debe formar parte de otro caso de uso base y no se presenta de forma aislada.
- Debe figurar en la especificación de todos los casos de uso que lo incluyan.

Como ejemplo de la relación de inclusión en diagramas de casos de uso, la Figura 24 muestra un diagrama que ilustra esta relación y en la Figura 25 presenta las tablas con la especificación detallada de estos casos de uso, destacando el caso incluido en el flujo principal del caso de uso.

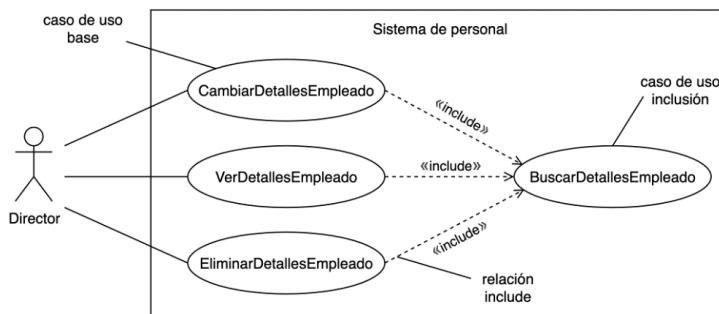


Figura 24: Ejemplo diagrama de casos de uso con inclusión [4]

Caso de uso: CambiarDetallesEmpleado	Caso de uso: VerDetallesEmpleado	Caso de uso: EliminarDetallesEmpleado
ID: 1	ID: 2	ID: 3
Breve descripción: El Director cambia los detalles del empleado.	Breve descripción: El Director ve los detalles del empleado.	Breve descripción: El Director elimina los detalles del empleado.
Actores principales: Director	Actores principales: Director	Actores principales: Director
Actores secundarios: Ninguno	Actores secundarios: Ninguno	Actores secundarios: Ninguno
Precondiciones: 1. El Director se conecta al sistema	Precondiciones: 1. El Director se conecta al sistema	Precondiciones: 1. El Director se conecta al sistema
Flujo principal: 1. include(BuscarDetallesEmpleado) 2. El sistema muestra los detalles del empleado. 3. El Director cambia los detalles del empleado. ...	Flujo principal: 1. include(BuscarDetallesEmpleado) 2. El sistema muestra los detalles del empleado. ...	Flujo principal: 1. include(BuscarDetallesEmpleado) 2. El sistema muestra los detalles del empleado. 3. El Director elimina los detalles del empleado. ...
Postcondiciones: 1. Se han cambiado los detalles del empleado.	Postcondiciones: 1. El sistema ha mostrado los detalles del empleado.	Postcondiciones: 1. Los detalles del empleado se han eliminado.
Flujos alternativos: Ninguno	Flujos alternativos: Ninguno	Flujos alternativos: Ninguno

Figura 25: Especificación de casos de uso con inclusión [4]

4. Extensión

- **Descripción:** Un caso de uso origen añade un comportamiento opcional al caso de uso destino, permitiendo modelar subflujos o características opcionales.
- **Independencia:** A diferencia de la inclusión, el caso de uso destino es funcional por sí mismo sin el caso de uso extendido.
- **Condiciones:** Útil para representar comportamientos que se activan solo bajo ciertas condiciones.
- **Representación:** La relación de extensión se indica con la etiqueta «extend».
- **Especificación:** La extensión debe mencionarse en la especificación de los casos de uso afectados, indicando los puntos donde se extiende el flujo principal.

Como ejemplo de la relación de extensión en un diagrama de casos de uso, la Figura 26 muestra esta relación, mientras que la Figura 27 presenta el caso de uso más detallado en el diagrama, indicando el punto de extensión tanto en el elemento como en su especificación.

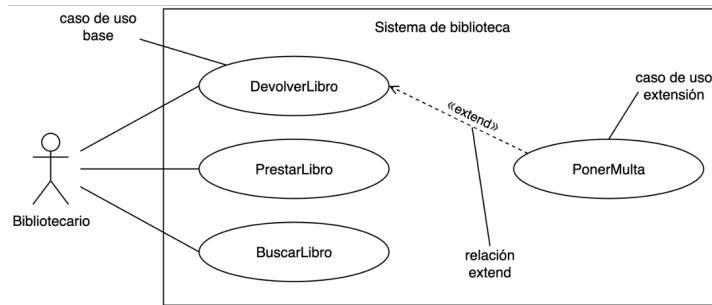


Figura 26: Ejemplo diagrama de casos de uso con extensión [4]

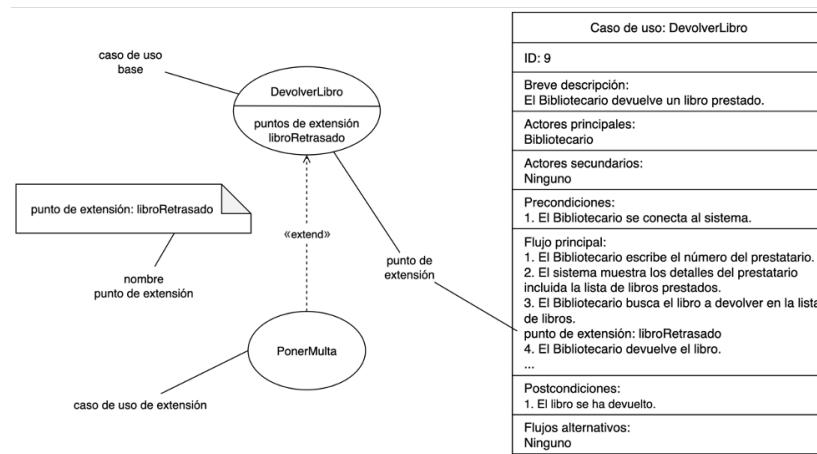


Figura 27: Especificación de casos de uso con extensión [4]

Trazabilidad

El modelo de casos de uso está estrechamente relacionado con la especificación de requisitos, un aspecto clave en la ingeniería de requisitos.

Es fundamental vincular los requisitos funcionales con los casos de uso para garantizar que:

- Cada **requisito funcional** está cubierto por al menos un caso de uso
- Cada **caso de uso** esté asociado a al menos un requisito funcional

Algunas herramientas de modelado o de gestión de requisitos permiten establecer vínculos automáticos entre los casos de uso y los requisitos.

Como alternativa, se puede crear una **matriz de trazabilidad** manual, con beneficios como:

- Detección de inconsistencias, como requisitos sin casos de uso o casos de uso sin requisitos asociados.
- Facilitación del seguimiento de los requisitos a lo largo del proyecto.

Para utilizar matrices de trazabilidad de manera efectiva, es importante que tanto los requisitos como los casos de uso tengan un ID único.

2.2. Diagrama de Actividades

Un **diagrama de actividades** modela un proceso como una secuencia de nodos conectados, de manera similar a un diagrama de flujo, pero orientado a objetos. Es una herramienta versátil en el modelado de sistemas que ofrece una representación visual clara de distintos tipos de flujos de trabajo.

Principales Usos

- Modelar visualmente el flujo de un caso de uso: Permite representar las secuencias de acciones en un caso de uso, facilitando su comprensión.
- Representar detalles de operaciones o algoritmos: Desglosa y documenta el comportamiento específico de algoritmos o métodos.
- Flexibilidad para diferentes tipos de flujos: Permite diagramar tanto flujos secuenciales como condicionales o paralelos.

La **actividad** es el elemento principal en un diagrama de actividades y se asocia a un elemento específico, como un caso de uso, clase o interfaz, con el fin de explicar su comportamiento. La actividad define el contexto del flujo de trabajo y puede hacer referencia a propiedades u operaciones del elemento asociado, proporcionando así mayor precisión y claridad en la modelación del proceso.

Las actividades en un diagrama de actividades se representan como una red de nodos conectados, donde cada tipo de nodo cumple una función específica. Existen tres tipos de nodos:

- **Nodos de Acción:** Representan unidades atómicas dentro de la actividad, que llevan a cabo acciones específicas.
- **Nodos de Control:** Gestionan y dirigen el flujo de la actividad, determinando cómo se transiciona entre nodos.
- **Nodos de Objeto:** Representan los objetos utilizados dentro de la actividad, facilitando la visualización de los recursos involucrados.

Además, existen dos tipos de flujo dentro de un diagrama de actividades:

- **Flujos de Control:** Conectan los nodos para establecer el orden en que se ejecutan las actividades.
- **Flujos de Objeto:** Conectan los nodos de objeto, representando la interacción y el intercambio de datos entre ellos.

Las actividades y sus nodos pueden tener precondiciones y postcondiciones, que establecen los requisitos que deben cumplirse antes de iniciar la actividad y los resultados esperados una vez finalizada.

En la Figura 28 se muestra la representación gráfica de los tres tipos de nodos en un diagrama de actividades. Los nodos de control se dividen en varias categorías, cada una con una representación específica. Por ejemplo, los nodos de inicio y fin de la actividad están marcados con símbolos específicos, lo que facilita la comprensión del flujo general del proceso.

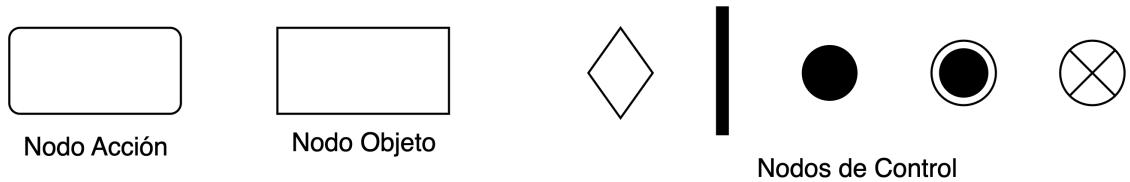


Figura 28: Tipos de nodos en un Diagrama de Actividades

En la Figura 29 se presenta un ejemplo de un diagrama de actividades que ilustra el proceso de realizar un pedido.

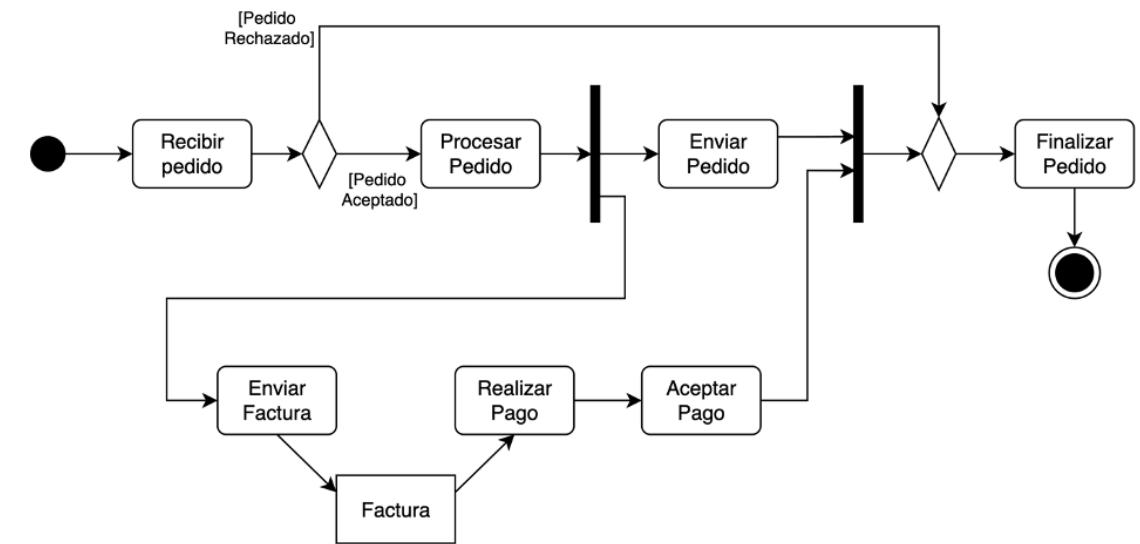


Figura 29: Ejemplo Diagrama de Actividades

Las **transiciones** conectan un estado de actividad o acción con el siguiente, permitiendo la representación del flujo dentro del diagrama. Existen varios tipos de transiciones:

- **Transiciones Secuenciales:** Representan un flujo simple entre actividades, donde cada actividad sigue a la anterior de manera directa.
- **Bifurcaciones:** Definen caminos alternativos en el flujo, determinados por expresiones booleanas.
 - Una bifurcación cuenta con una transición de entrada y dos o más transiciones de salida.
 - Cada transición de salida está asociada a una condición booleana que se evalúa una sola vez.
 - Es importante que las condiciones en cada salida no se solapen y que cubran todas las opciones posibles para garantizar un flujo continuo y eficiente.

Nodos de Control

Los **nodos de control** administran y dirigen el flujo de actividades, estableciendo cómo se conectan y organizan los distintos elementos en el diagrama.

Tipos de Nodos de Control

- **Nodos de Inicio:** Permiten iniciar uno o más flujos de manera concurrente dentro de la actividad. Su representación gráfica es un círculo de color negro (Figura 28).
- **Nodo de Fin:** Puede detener todos los flujos activos de la actividad o solo uno, según el contexto. Su representación gráfica es un círculo de color negro dentro de otro círculo blanco (Figura 28).
- **Nodos de Decisión:** Tienen un único punto de entrada y varios puntos de salida, cada uno con una condición exclusiva que dirige el flujo hacia diferentes caminos. Se suele representar como un rombo con una condición (Figura 30a).
- **Nodos de Fusión:** Permiten unir varios flujos de entrada en un único flujo de salida, consolidando diferentes trayectorias en un solo flujo. Se representa igual que el de decisión (Figura 30a).
- **Nodo “Fork”:** Divide un flujo en varios subflujos paralelos, permitiendo que múltiples actividades se ejecuten simultáneamente. Se representa con una línea continua gruesa que recibe un flujo y devuelve dos o más (Figura 30b).

- **Nodo “Join”:** Sincroniza los flujos paralelos generados, combinándolos en un único flujo de salida. Al igual que el “Fork” es una línea continua que recibe más de un flujo y retorna solo uno (Figura 30b).

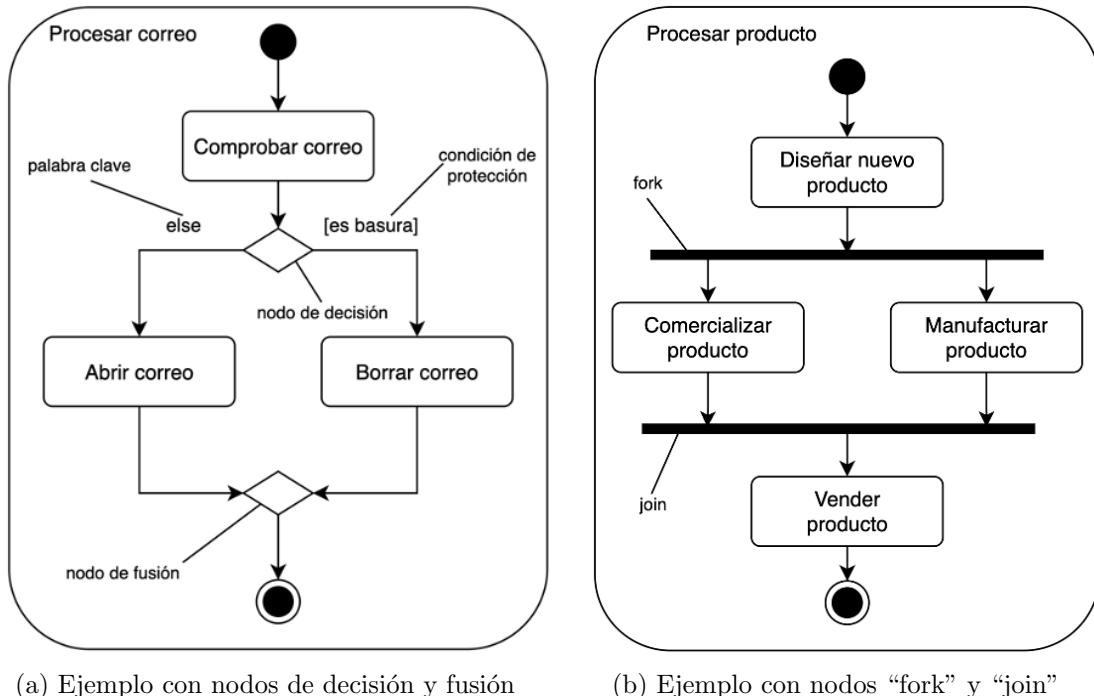


Figura 30: Ejemplos con nodos de control [4]

Nodos de Objeto

Los **nodos de objeto** indican la disponibilidad de una instancia en un punto específico de la actividad. Estos nodos:

- Tienen flujos de entrada y salida de tipo objeto.
- Representan la creación y el uso de objetos a través de los nodos de acción, permitiendo visualizar cómo y cuándo se manipulan los objetos dentro del flujo de la actividad.

Particiones de Actividad

Las particiones de actividad, también conocidas como carriles o **calles** (*swim-lanes*), dividen el diagrama de actividades horizontal o verticalmente para mejorar su claridad. Cada partición lleva un nombre único y agrupa acciones relacionadas, facilitando la asociación de acciones con diferentes elementos o responsables. La interpretación de estas particiones puede variar según el criterio del modelador, ofreciendo flexibilidad en la organización del flujo.

En la Figura 31 se muestra un ejemplo de un diagrama de actividades dividido en calles, y en la Figura 32 se presenta una comparación de otro ejemplo de diagrama sin y con calles, para visualizar el efecto de esta organización.

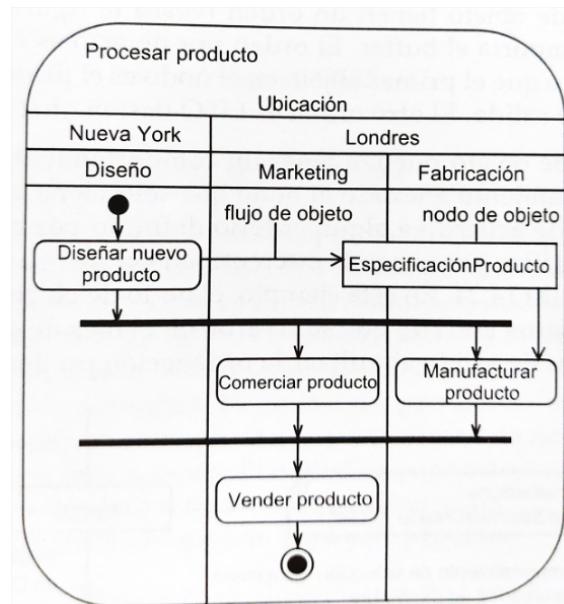


Figura 31: Diagrama de Actividades estructurado por calles [4]

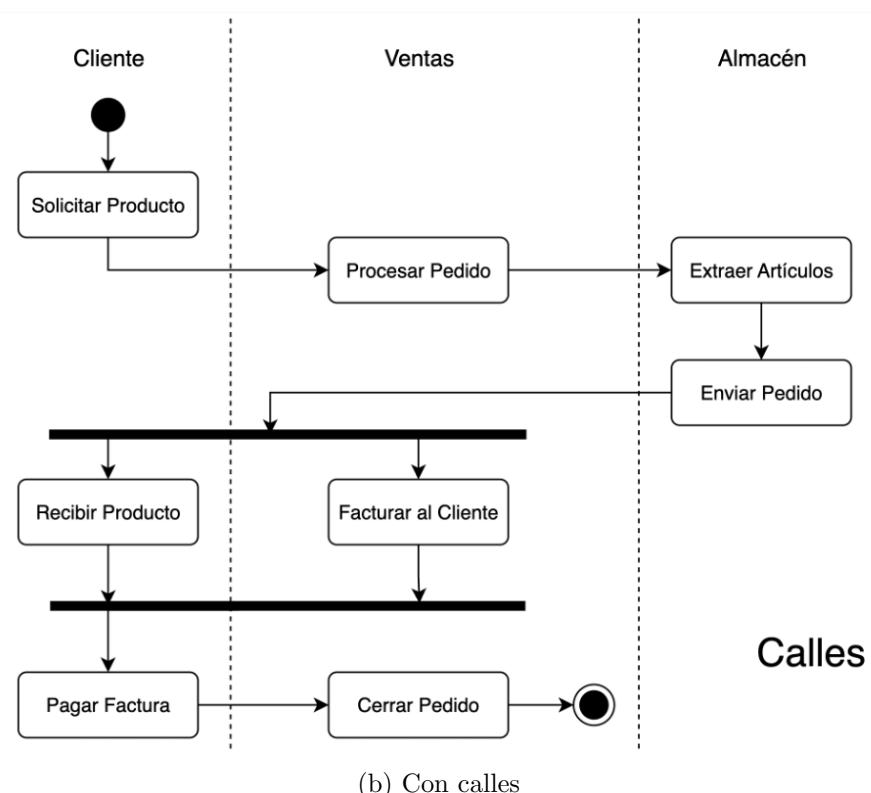
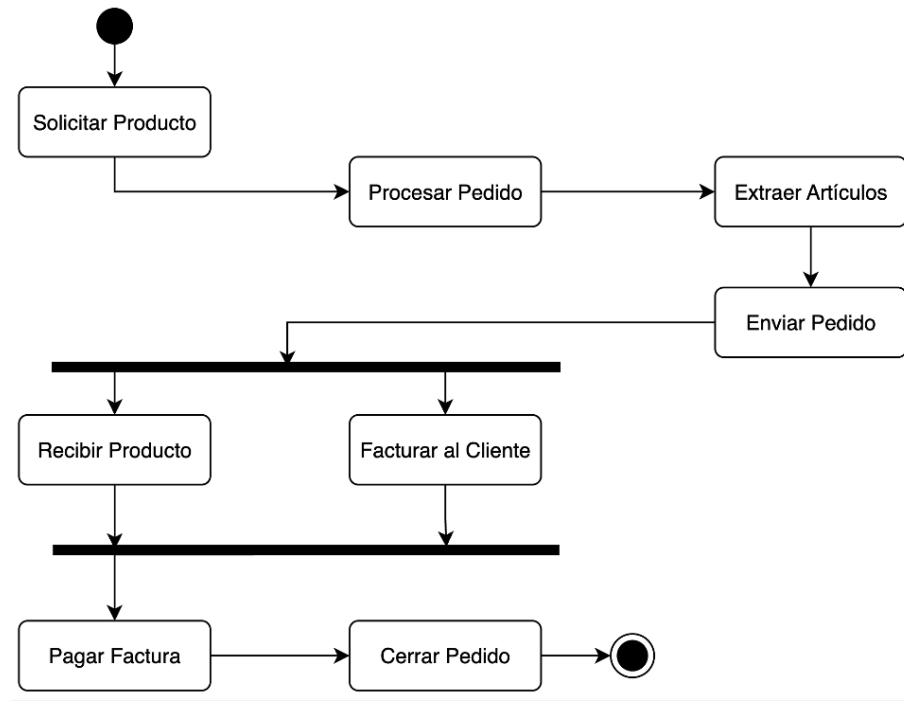


Figura 32: Ejemplo Diagrama de Actividad

2.3. Diagrama de Máquinas de Estado

Una **máquina de estados** define la secuencia de estados que un objeto atravesía a lo largo de su vida, impulsada por eventos internos o externos. Este tipo de diagrama es útil para modelar el comportamiento dinámico de elementos como clases, casos de uso o sistemas completos.

Relación con Otros Diagramas:

- **Diagramas de Interacción:** Estos diagramas representan el comportamiento de múltiples objetos interactuando, mientras que la máquina de estados modela el comportamiento de un solo objeto.
- **Diagramas de Actividades:** Se centran en el flujo de control entre actividades, no en los estados. En estos diagramas, la transición entre actividades se da al completarse la actividad anterior.

Elementos de una Máquina de Estados

1. **Estados:** Representan las condiciones o situaciones en las que se encuentra un objeto en un momento dado.
2. **Transiciones:** Indican las posibles rutas de cambio entre estados.
3. **Eventos:** Son sucesos que desencadenan cambios de estado en un momento específico.

En el diagrama, los estados de inicio y fin tienen una representación visual distinta de los demás estados, lo que ayuda a identificar claramente el inicio y la finalización del flujo.

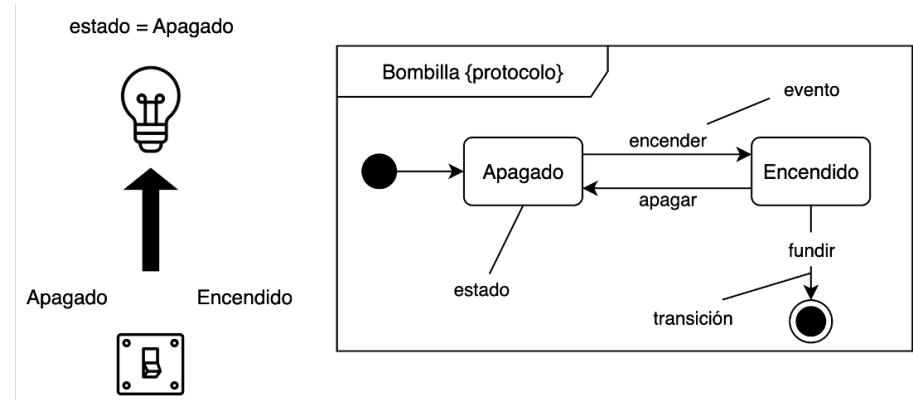


Figura 33: Ejemplo diagrama de máquina de estados de una bombilla

Estados

Un **estado** en una máquina de estados representa una condición o situación en la vida de un objeto. En este estado, el objeto puede cumplir alguna condición específica, realizar una actividad, o estar en espera de un evento que pueda desencadenar un cambio. Los estados permiten modelar cómo un objeto responde y se adapta a diferentes circunstancias a lo largo de su ciclo de vida. Un objeto permanece en cada estado durante un período finito antes de realizar una transición a otro estado en respuesta a eventos.

Partes de un Estado

- **Nombre:** Texto que identifica el estado; en algunos casos, puede no ser especificado.
- **Acciones de Entrada/Salida:** Acciones que se ejecutan al ingresar y al salir del estado.
- **Transiciones Internas:** Cambios que se manejan dentro del estado sin necesidad de salir de él.
- **Actividades Internas:** Operaciones que requieren un tiempo para completarse y que pueden interrumpirse si ocurre un evento.

También pueden existir **estados compuestos**; estados que contienen otros estados anidados en su interior, conocidos como submáquinas. Los estados anidados heredan todas las transiciones de sus estados contenedores, lo que facilita la organización y modularidad en el diagrama de estados. Pueden clasificarse en:

- Sencillos: Cuentan con una sola región para los estados anidados.
- Ortogonales: Tienen múltiples regiones, permitiendo el modelado de comportamientos concurrentes.

Los eventos compuestos se pueden anidar a distintos niveles, aunque se recomienda limitar esta profundidad a dos o tres niveles para mantener la claridad y comprensión del modelo.

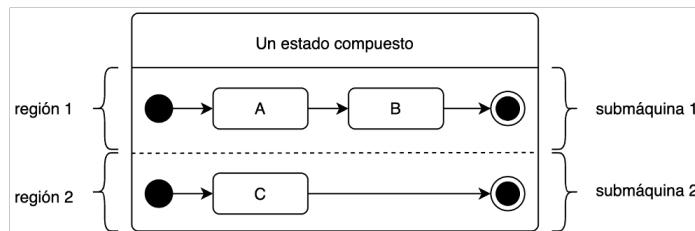


Figura 34: Ejemplo de un estado compuesto

Transiciones

Una **transición** es la relación entre dos estados que indica que un objeto ejecutará ciertas acciones y cambiará al segundo estado cuando ocurra un evento específico y se cumplan las condiciones necesarias. Este proceso de cambio es fundamental para modelar la respuesta de un objeto ante diversos eventos y su evolución a lo largo del ciclo de vida del sistema.

Cuando se produce el cambio de un estado a otro, se dice que la **transición se ha disparado**. Antes de dispararse, el objeto se encuentra en el estado de origen; una vez disparada, pasa al estado de destino. En los diagramas, una transición se representa mediante una línea continua dirigida desde el estado de origen hacia el estado de destino, lo que ayuda a visualizar el flujo de comportamiento del objeto.

Un caso especial de transición es la **auto-transición**, donde el estado de origen y el de destino son el mismo. Esta transición es útil para representar cambios internos o actividades dentro de un estado sin pasar a un estado diferente. Además, en una máquina de estados puede haber transiciones con múltiples orígenes (denominadas **join** o unión de estados concurrentes) o con múltiples destinos (denominadas **fork** o división hacia varios estados concurrentes). Estas transiciones permiten modelar comportamientos complejos y situaciones donde el objeto puede dividirse o sincronizarse en varios flujos paralelos de actividades.

Una transición puede incluir tres elementos opcionales:

- **Evento(s) de disparo:** Son las ocurrencias internas o externas que activan la transición y desencadenan el cambio de estado del objeto.
- **Condición(es) de guarda:** También conocidas como condiciones de protección, son expresiones booleanas escritas entre corchetes que se evalúan una vez que ocurre el evento de disparo. Solo si esta condición es verdadera, la transición procederá.
- **Acción(es):** Son los procesos a ejecutar cuando se activa la transición. Estas acciones pueden incluir llamadas a operaciones sobre el objeto, cambios en sus propiedades, entre otras modificaciones. Una vez iniciada, la acción no puede ser interrumpida y se ejecuta hasta completarse.

En la Figura 35 se puede ver un ejemplo de los tres elementos de una transición.

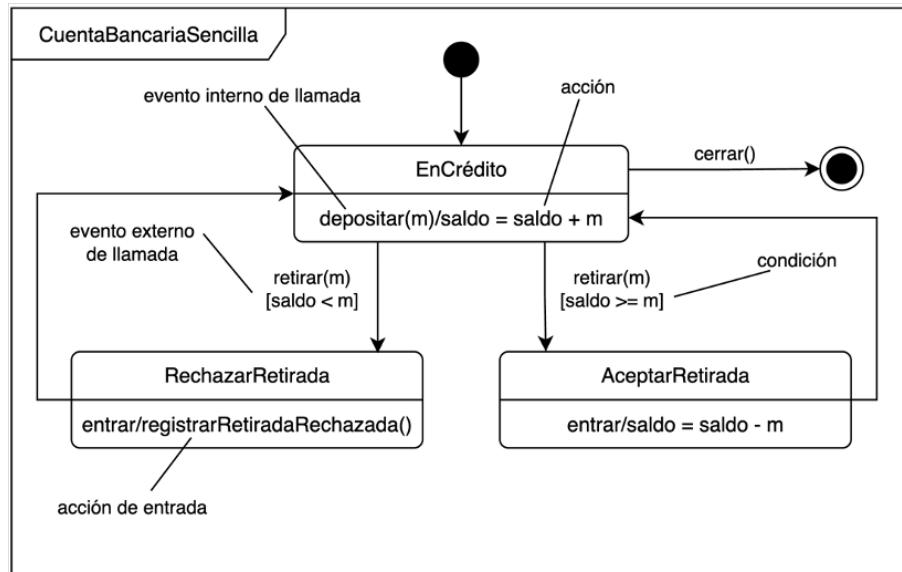


Figura 35: Ejemplo diagrama máquina de estados de una cuenta bancaria

Eventos

Un **evento** es un acontecimiento significativo que ocurre en un momento y lugar específicos. En las máquinas de estados, los eventos modelan los estímulos que desencadenan un cambio de estado. Estos eventos pueden ocurrir de forma externa (representados sobre la transición) o interna (dentro de un estado).

Tipos de Eventos

- **Evento de Llamada:** Representa la solicitud de una operación específica en un objeto. Cuando una operación se invoca sobre un objeto con una máquina de estados, el proceso sigue estos pasos:
 - El control pasa del emisor al receptor.
 - El evento activa la transición y la operación se ejecuta.
 - El receptor cambia a un nuevo estado, tras lo cual el control vuelve al emisor.
 - Para asegurar coherencia, la operación debe estar incluida en la lista de operaciones del objeto receptor y debe coincidir con la firma esperada.
- **Evento de Señal:** Consiste en el envío asíncrono de información sin asociarla a una operación específica. Este tipo de evento tiene una representación especial tanto para el envío como para la recepción de señales.

- **Evento de Tiempo:** Representa el paso del tiempo y se modela con la palabra “after” seguida de una expresión que especifica el intervalo temporal. El tiempo se mide a partir del momento en que el objeto entra en el estado actual.
- **Evento de Cambio:** Este evento representa una modificación en el estado o el cumplimiento de una condición y se modela con la palabra “when” seguida de una expresión booleana.

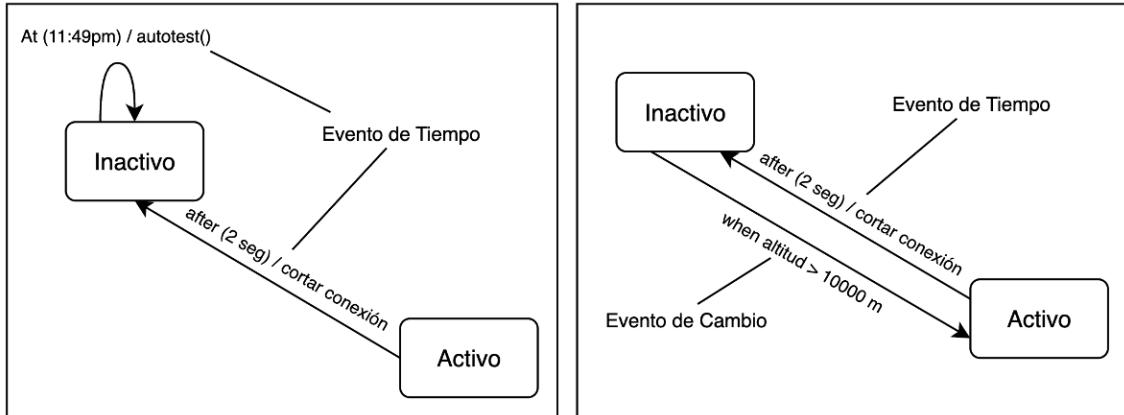


Figura 36: Ejemplos de eventos de tiempo y cambio

Finalmente, en la Figura 37 se puede observar un ejemplo más completo de un diagrama de máquina de estados.

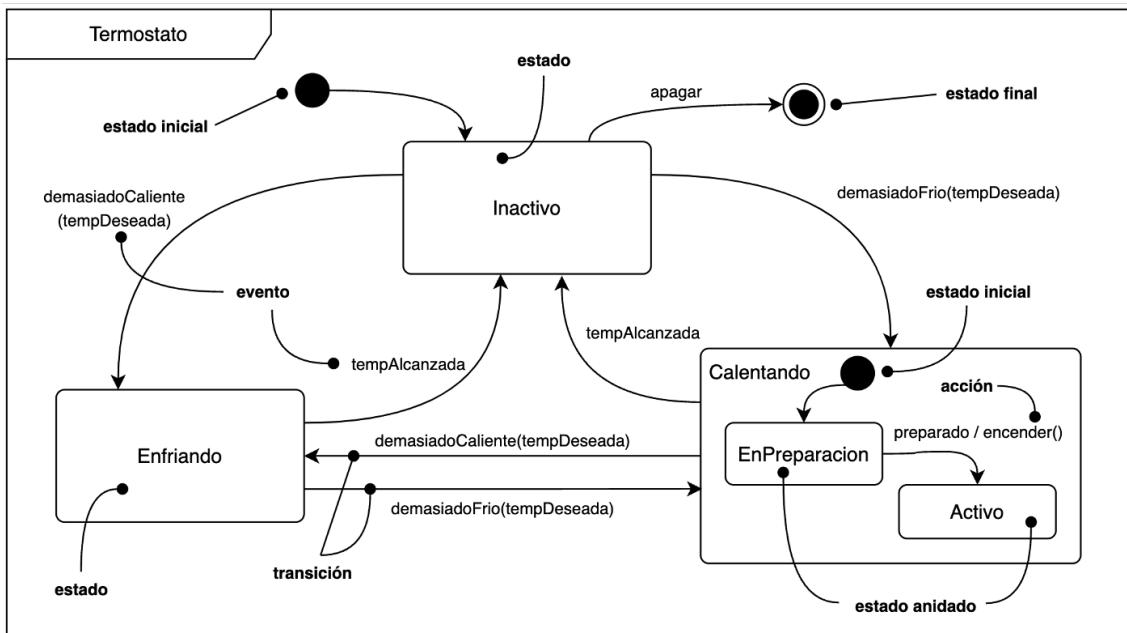


Figura 37: Ejemplo diagrama máquina de estados de un termostato

2.4. Diagrama de Interacción

Un **diagrama de interacción** representa el comportamiento dinámico del sistema y el flujo de control durante una operación. Describen cómo los objetos interactúan entre sí mediante el intercambio de mensajes para llevar a cabo tareas específicas. Estas interacciones modelan un “comportamiento” y ayudan a “realizar” o implementar un caso de uso.

Los elementos principales en los diagramas de interacción son:

- **Objetos:** participan en la interacción.
- **Enlaces:** establecen conexiones entre los objetos.
- **Mensajes:** representan la comunicación entre objetos.

Además, los diagramas de interacción pueden incluir notas y restricciones para mayor claridad en la modelación.

Existen cuatro tipos de diagramas de interacción, pero en esta sección nos centraremos en dos:

- **Diagrama de Secuencia:** Enfatiza el orden temporal de los eventos. Visualmente, se representa como una tabla donde los objetos se disponen en el eje horizontal (X) y los mensajes en el eje vertical (Y), con los eventos ordenados cronológicamente.
- **Diagrama de Colaboración:** Resalta la estructura y las relaciones entre los objetos que envían y reciben mensajes. Gráficamente, aparece como una red de nodos y arcos que muestra cómo los objetos están conectados.

Ambos diagramas son semánticamente equivalentes, lo que significa que se puede convertir uno en el otro sin perder información. Sin embargo, visualmente, cada tipo destaca diferentes aspectos de la interacción, por lo que algunos detalles pueden percibirse con más claridad en uno que en el otro.

Estos diagramas de interacción reflejan cómo los objetos colaboran para realizar las funciones de una aplicación, representando la comunicación entre ellos en una tarea compartida.

Tipo	Ventajas	Desventajas
Diagrama de Secuencia	<ul style="list-style-type: none"> - Notación simple - Ilustra mejor las secuencias 	<ul style="list-style-type: none"> - Elaboración rígida
Diagrama de Colaboración	<ul style="list-style-type: none"> - Elaboración flexible - Ilustra mejor condicionales, iteraciones y concurrencias 	<ul style="list-style-type: none"> - Notación compleja - Ilustra peor las secuencias

Tabla 2: Comparación de Tipos de Diagramas de Interacción

Una **interacción** se define como una unidad de comportamiento de un clasificador, como un caso de uso, que ilustra cómo se lleva a cabo la comunicación a través del intercambio de mensajes. Esta unidad de comportamiento muestra cómo los objetos colaboran para cumplir con un objetivo específico, reflejando el flujo de control y las dinámicas entre ellos.

Por otro lado, un **mensaje** representa un tipo específico de comunicación dentro de una interacción. Los mensajes pueden invocar operaciones, crear o destruir instancias, o enviar señales entre los objetos. En este contexto, los mensajes son los elementos que facilitan la interacción y permiten que los objetos se comuniquen y colaboren entre sí.

Las interacciones, junto con sus mensajes, son fundamentales para modelar el comportamiento dinámico de las colaboraciones, que pueden incluir clases, interfaces, componentes, nodos y casos de uso. Estos aspectos dinámicos se representan mediante flujos de control que pueden variar desde secuencias simples hasta flujos más complejos que incluyen bifurcaciones, iteraciones, recursión y concurrencia.

En la Figura 38 se ilustra la representación gráfica de una clase, una instancia y una instancia nombrada en el contexto de un diagrama de interacción. La **clase** define los atributos y métodos comunes que pueden compartir sus instancias, sirviendo como un modelo para la creación de objetos. La **instancia** representa un objeto concreto creado a partir de esa clase, mientras que la **instancia nombrada** se refiere a una instancia específica que se identifica con un nombre único dentro del diagrama.

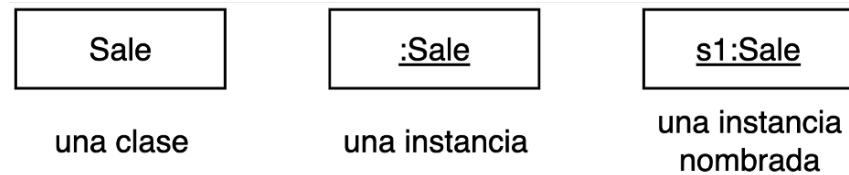


Figura 38: Notación gráfica de clases e instancias

Esta notación es fundamental para visualizar las interacciones entre objetos, mostrando cómo se comunican y colaboran para llevar a cabo funciones en el sistema.

En los diagramas de interacción se representa la definición de un mensaje que se envía entre objetos con la siguiente sintaxis:

```
return := nombreMensaje(parametro : tipoParam,...) : tipoRetorno
```

En la Tabla 3 se muestran todos los tipos de mensajes de un diagrama de interacción y su representación gráfica.

Nombre	Notación	Significado
Mensaje síncrono	→	El emisor espera hasta recibir respuesta del receptor
Mensaje asíncrono	→	El emisor prosigue su ejecución sin esperar respuesta del receptor
Retorno de mensaje	<-----	Devolución del foco de control
Creación de objeto	→ <small><<create>></small>	Creación de una instancia
Destrucción de objeto	→ ✘	Destrucción de una instancia
Mensaje encontrado	● →	Llegada de un mensaje sin especificar emisor
Mensaje perdido	→ ●	Mensaje que no llega a su destino (error)

Tabla 3: Notación de Mensajes en Diagramas de Interacción

La **secuenciación** se refiere a la dinámica en la que, cuando un objeto envía un mensaje a otro, el receptor puede comunicarse a su vez con un tercer objeto, lo que desencadena una serie de mensajes en forma de secuencia. Esta secuencia comienza con un proceso o hilo y se mantiene activa mientras dicho proceso o hilo exista. Cada proceso o hilo establece un flujo de control independiente, organizando los mensajes en un orden cronológico. Para facilitar la visualización de esta secuencia, se utiliza un número de secuencia, como 1, 1.1, 1.2, etc., que indica el orden de los mensajes desde el inicio de la interacción.

2.4.1. Diagrama de Secuencia

Un **diagrama de secuencia** representan la interacción entre los componentes del sistema desde una perspectiva temporal. La interacción se ilustra mediante el flujo de mensajes entre objetos o actores a lo largo del tiempo.

Su utilidad radica en la capacidad de describir procesos internos entre diversos módulos y en detallar las comunicaciones tanto con otros sistemas como con actores externos. A menudo, estos diagramas se vinculan a los casos de uso, mostrando cómo se llevan a cabo a través de interacciones entre diferentes entidades de objetos.

En la Tabla 4 se pueden ver la representación gráfica de los principales elementos en un diagrama de secuencia

Nombre	Notación
Actor	:Actor
Objeto	Objeto:Clase
Foco de control	.
Actores/objetos desconocidos	.
Fin de una "línea de vida"	X

Tabla 4: Notación de los elementos en un diagrama de secuencia

Cada actor u objeto está representado por un eje vertical, conocido como línea de vida, que ilustra su existencia a lo largo del tiempo.

El intercambio de mensajes se indica mediante líneas horizontales que conectan los objetos, junto con la descripción correspondiente de cada mensaje.

Cuando un objeto o actor está activo, se representa mediante un rectángulo sobre la línea de tiempo, cuyo tamaño corresponde a la duración de su actividad.

En las Figuras 39 y 40 se pueden ver dos ejemplos de diagrama de secuencia.

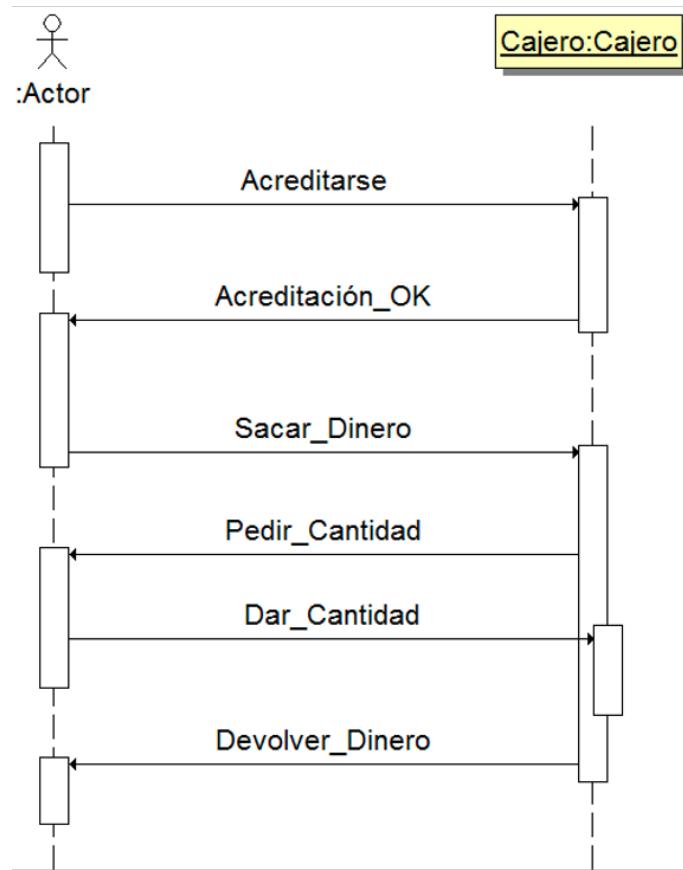


Figura 39: Ejemplo Diagrama de Secuencia

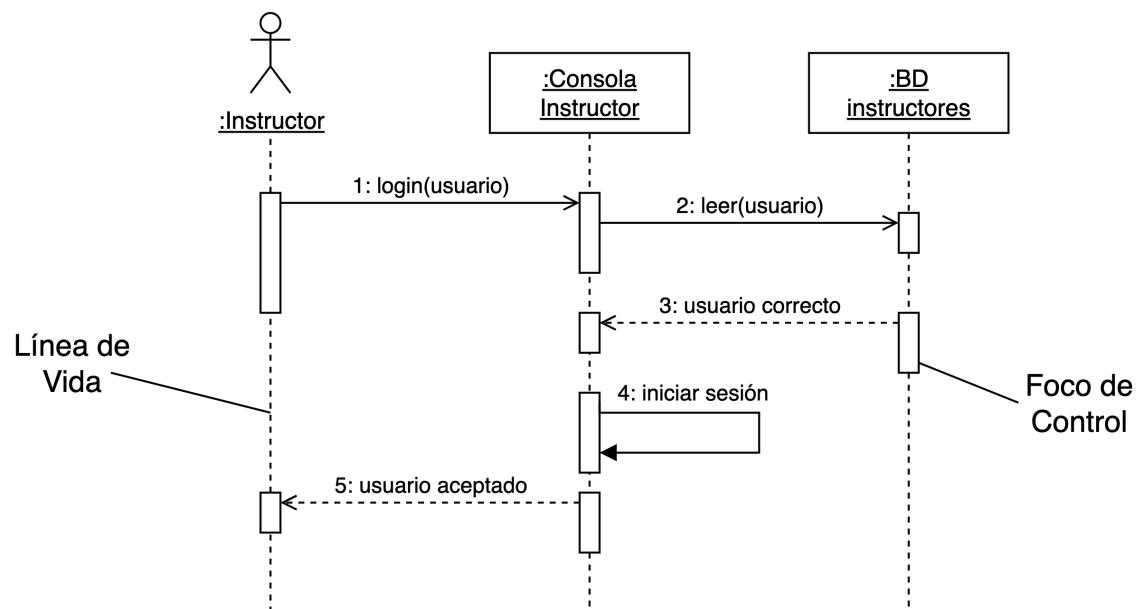


Figura 40: Ejemplo Diagrama de Secuencia

Los diagramas de secuencia se distinguen de los diagramas de colaboración por dos características principales:

- **Línea de vida:** Se representa como una línea vertical discontinua y señala la duración de un objeto. Esta línea comienza con el mensaje “create” y finaliza con “destroy”, incluyendo una señal en forma de X al final.
- **Foco de control:** Se representa como un rectángulo delgado y estrecho sobre la línea de vida. Este foco de control indica el tiempo de ejecución de una acción y permite la posibilidad de anidar rectángulos adicionales hacia la derecha.

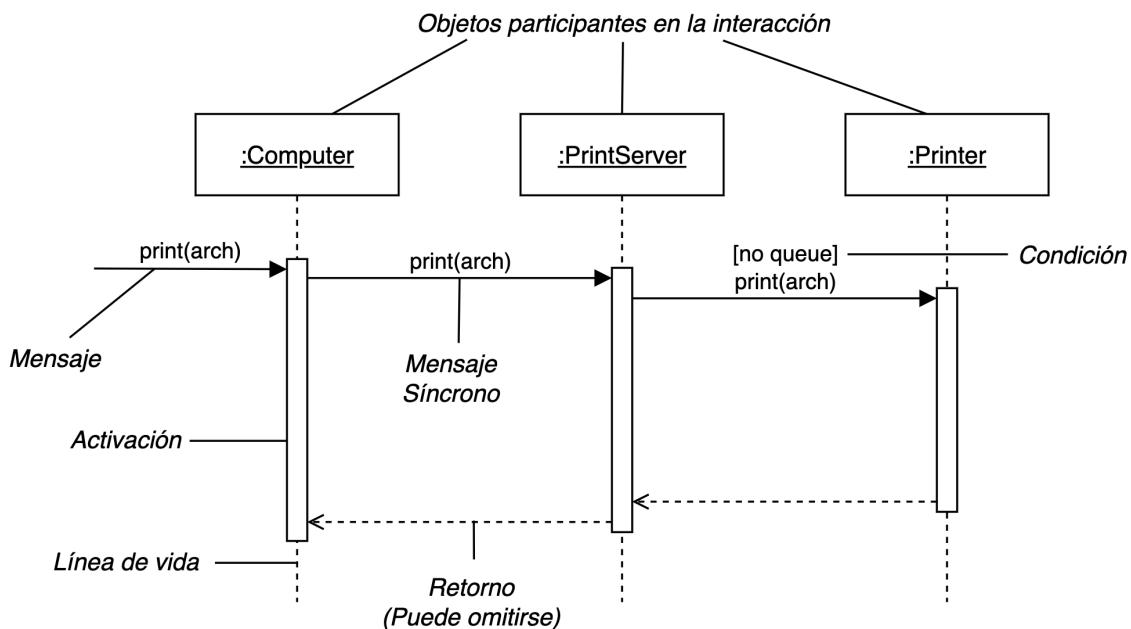


Figura 41: Ejemplo Diagrama de Secuencia

Operadores de Control y Marcos de Interacción

Un **marco de interacción** es una sección dentro de un diagrama de secuencia que incluye un operador de control, el cual define cómo se ejecuta la secuencia de mensajes. A continuación se describen los principales tipos de ejecución:

- **Ejecución Opcional (opt):** Este operador se ejecuta solo si se cumple una condición específica (Figura 42a).
- **Ejecución Condicional (alt):** En este caso, el operador se divide en varias subregiones, cada una representando una rama de la condición. Las subregiones se delimitan con líneas discontinuas horizontales (Figura 42b).

- **Ejecución Iterativa (loop):** Este operador indica un comportamiento repetitivo, especificando valores mínimos y máximos, así como la condición de guarda. El cuerpo del bucle se ejecuta mientras la condición de guarda sea verdadera antes de cada iteración (Figura 42c).
- **Ejecución Paralela (par):** Aquí, el operador de control se divide en varias subregiones separadas por líneas discontinuas horizontales. Cada subregión indica una computación paralela o concurrente (Figura 42d).

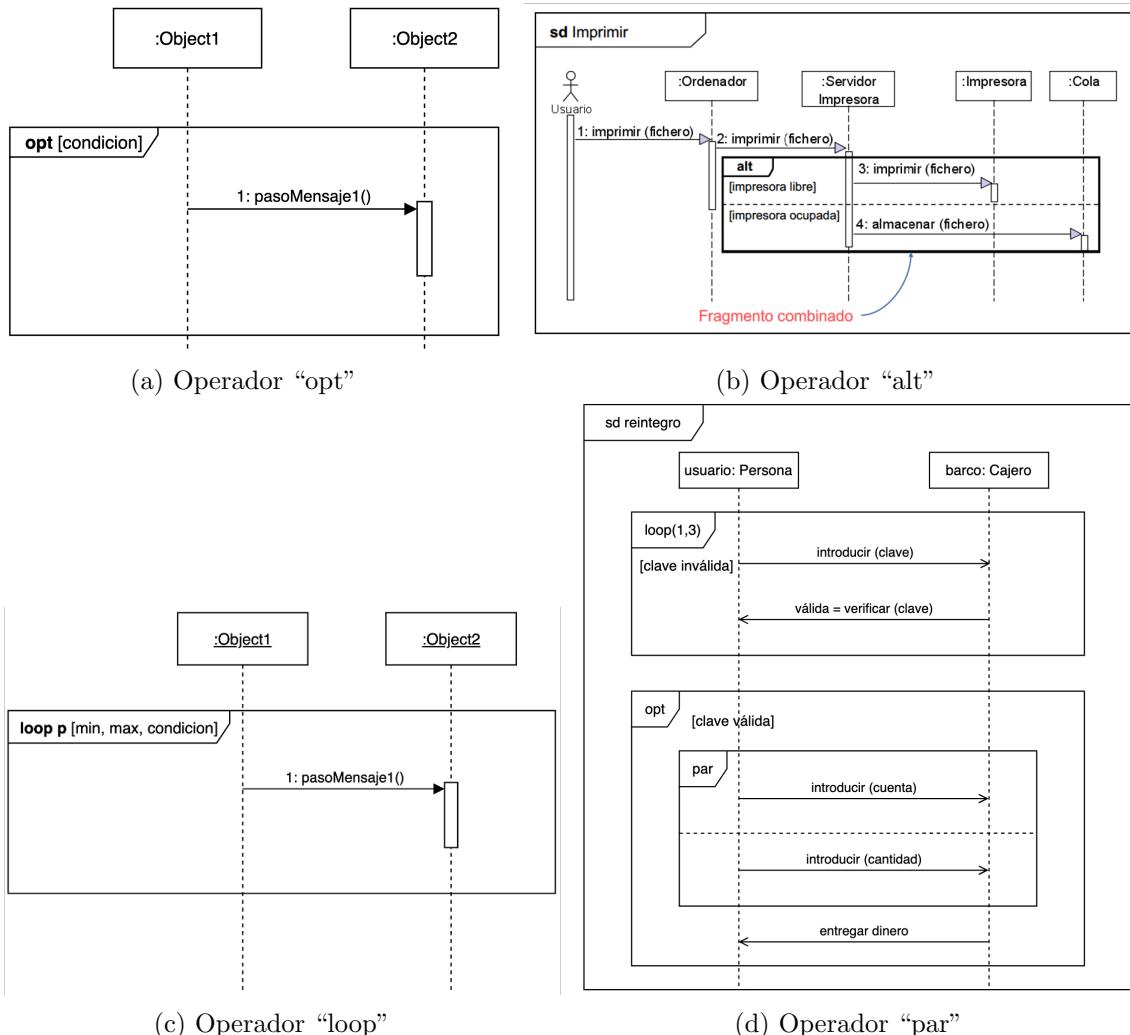


Figura 42: Operadores de Control

2.4.2. Diagrama de Colaboración

El **diagrama de colaboración** ilustra la interacción espacial entre objetos, centrándose en el intercambio de mensajes. Este tipo de diagrama utiliza los mismos elementos que los diagramas de secuencia, aunque carece de las “líneas de vida” y de ‘foco de control’.

En cuanto a su utilidad, el diagrama de colaboración permite identificar los objetos y sus relaciones dentro del sistema. También describe el flujo de mensajes entre los objetos o roles, resaltando la organización estructural de los objetos que colaboran. En este contexto, los objetos se representan como nodos, mientras que los enlaces se muestran como arcos etiquetados con los mensajes. Además, el orden de los mensajes se indica utilizando la numeración decimal de Dewey (1, 1.1, ...).

Los mensajes se representan mediante un enlace entre dos objetos, el cual se visualiza como una línea. Los mensajes se superponen a este enlace, y el orden de ejecución se indica junto a su descripción textual.

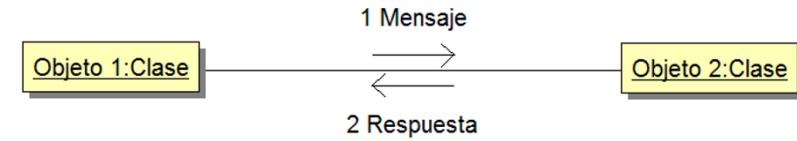


Figura 43: Mensajes. Diagrama de Colaboración

Cada mensaje puede expresarse en lenguaje natural o en pseudocódigo, lo que permite incluir condiciones o invocaciones a funciones.

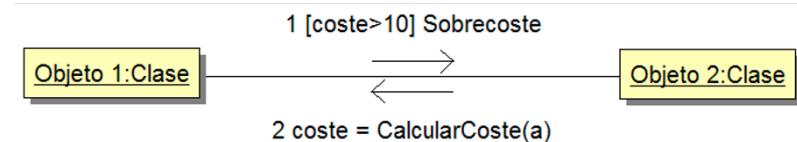


Figura 44: Mensajes. Diagrama de Colaboración

En la Figura 45 se puede ver un diagrama de colaboración para la función “ingresar un ítem” en una aplicación de Punto-de-Venta.

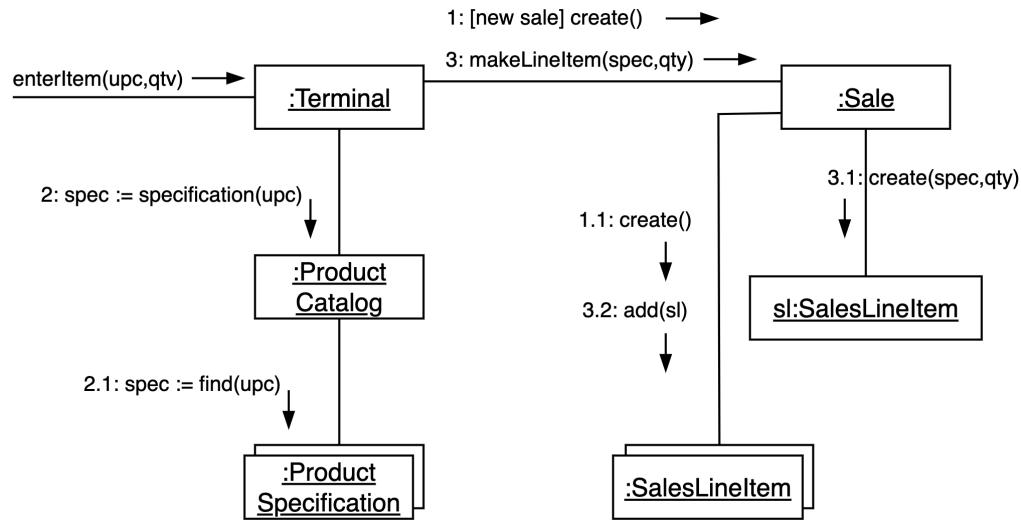


Figura 45: Ejemplo Diagrama de Colaboración

En la Figura 46 se muestra otro ejemplo, en este caso del préstamo de libros en una biblioteca.

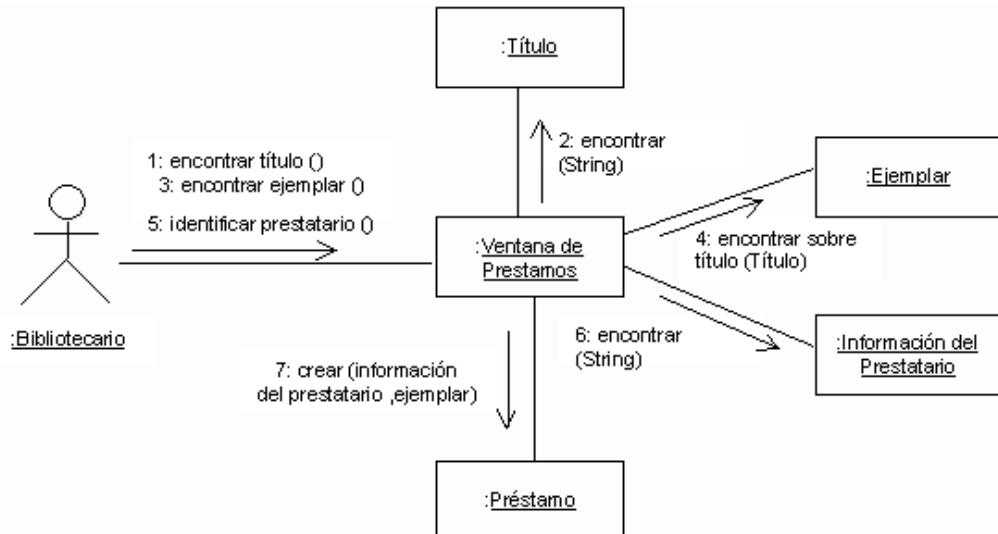


Figura 46: Ejemplo Diagrama de Colaboración

3. El Modelo Estructural en UML

En UML, las partes estáticas del sistema se representan mediante varios tipos de diagramas que permiten capturar su estructura fundamental:

- Diagrama de Clases
- Diagrama de Objetos
- Diagrama de Paquetes
- Diagrama de Componentes
- Diagrama de Despliegue

Cada uno de estos diagramas contribuye a una representación clara y organizada de los elementos que componen el sistema y sus relaciones internas.

Los diagramas estructurales son esenciales para visualizar, especificar, construir y documentar la estructura estable del sistema, que a menudo se denomina su “esqueleto”. Estos diagramas proporcionan una vista completa de la arquitectura estática del sistema, que permanece constante durante su operación.

Dentro de estos diagramas, se incluyen varios elementos clave como clases, interfaces, colaboraciones, componentes y nodos. Cada uno desempeña un papel específico en la definición de la estructura del sistema, ayudando a entender cómo se interrelacionan los distintos elementos del software para crear una base sólida sobre la cual se ejecutan las funcionalidades del sistema.

3.1. Diagrama de Clases

Una **clase** define una categoría de objetos que comparten ciertas características de estructura (atributos) y comportamiento (operaciones). Los **objetos** son instancias específicas de una clase y tienen valores concretos asignados a sus atributos.

Cada clase incluye una serie de operaciones que los objetos pueden ejecutar cuando son invocados. Los objetos se activan al ser creados y permanecen activos hasta que completan su función o son terminados por otro objeto en el sistema.

Mientras están activos, los objetos pueden recibir y responder a mensajes de otros objetos, lo que permite la interacción y colaboración entre ellos dentro del sistema.

Compartimentos de una Clase

La especificación de una clase se compone de varias secciones o compartimentos que permiten describir sus características en detalle:

1. Compartimento de nombre:

- Nombre de la clase (obligatorio).
- Estereotipos: pueden incluirse si es necesario, como ‘‘abstract’’ para indicar que la clase es abstracta.
- Valores etiquetados: permiten añadir metadatos adicionales a la clase.

2. Compartimento de atributos:

detalla cada atributo de la clase, incluyendo:

- Nombre del atributo (obligatorio).
- Visibilidad: puede ser pública (+), privada (-), protegida (#) o de paquete (~).
- Tipo de dato, donde UML soporta tipos primitivos como ‘int’, ‘string’, ‘boolean’ y enumeraciones.
- Multiplicidad: define el rango de instancias permitidas, como ‘[0...1]’, ‘[3]’, o ‘[2...*]’.
- Valor inicial (opcional): útil en la fase de diseño, principalmente para indicar restricciones.

3. Compartimento de operaciones:

lista cada operación que la clase puede realizar, indicando:

- Nombre de la operación (obligatorio, especialmente en fases tempranas de análisis).
- Visibilidad de la operación (pública, privada, protegida o de paquete).
- Tipo de retorno, el cual debe ajustarse a los tipos soportados en UML.
- Lista de parámetros (más comúnmente detallada en el diseño) con:
 - Dirección del parámetro (in, out, inout).
 - Tipo de cada parámetro.
 - Valor predeterminado, si aplica.

Es importante recordar que estos elementos pueden simplificarse o refinarse durante el proceso de análisis y diseño, y no siempre todos se presentan en el diagrama. Además, no tiene por qué haber una correspondencia exacta entre esta especificación y la sintaxis de un lenguaje de programación. Un ejemplo de propiedad adicional en una operación es ‘isQuery’, indicando que la operación no modifica el estado del objeto. En la Figura 47 se muestran los tipos de compartimentos.

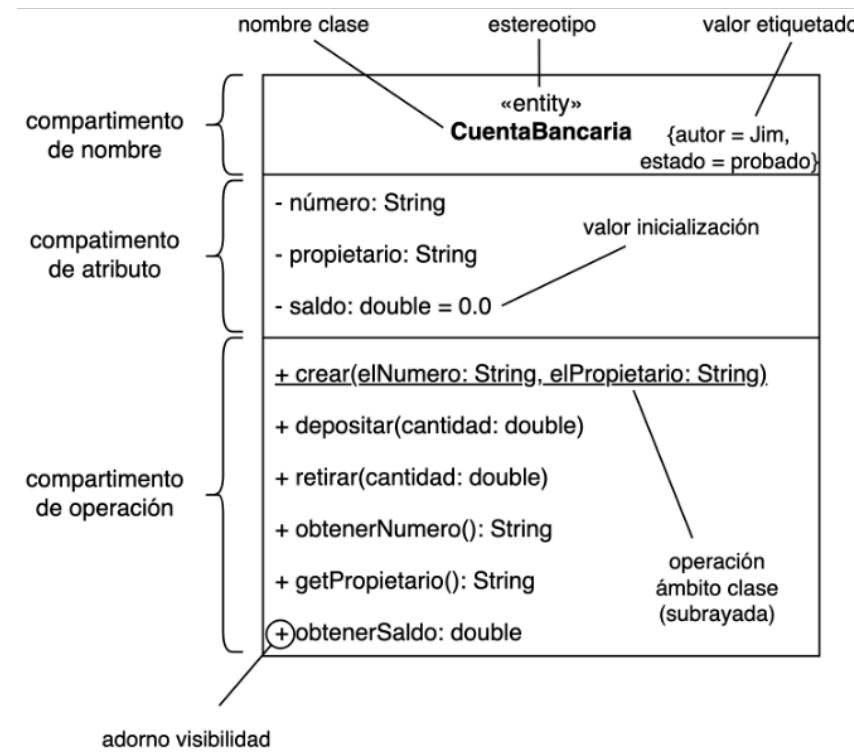


Figura 47: Compartimentos en una clase

Las clases pueden representarse de diversas maneras, según el nivel de detalle que se deseé incluir. En la Figura 48 se presentan distintos ejemplos de clases y la variedad de información que pueden mostrar.

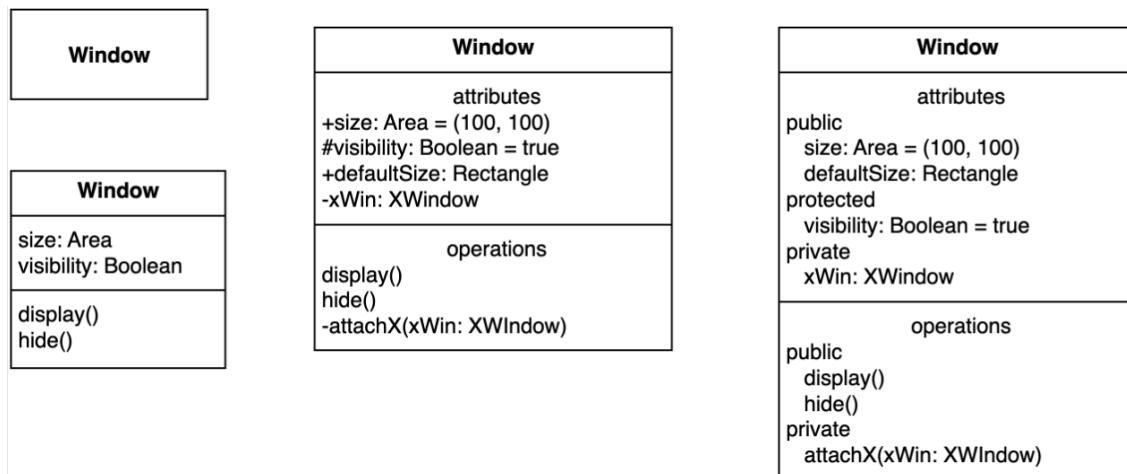


Figura 48: Ejemplos de representación de una clase

Relaciones

Las relaciones en un modelo representan conexiones semánticas significativas entre distintos elementos. En el contexto del modelado de clases y objetos, se pueden identificar tres tipos principales de relaciones:

1. **Relaciones entre objetos:** también llamadas vínculos, representa una conexión directa entre dos objetos, lo que indica que estos pueden intercambiar mensajes. Cuando un objeto recibe un mensaje, se ejecuta la operación correspondiente.
2. **Relaciones de dependencia:** expresa que existe una relación entre dos o más elementos, donde un cambio en uno de ellos (llamado el proveedor) podría afectar a otro (llamado el cliente). Un ejemplo común es cuando una operación de una clase tiene un parámetro cuyo tipo corresponde a otra clase.
Para especificar diferentes tipos de dependencias, se emplean diversos estereotipos que permiten ajustar la semántica de la relación según el contexto del modelo.
3. **Relaciones entre clases:**

- **Generalización:** es una relación entre un elemento general y una versión más específica de ese elemento. En la mayoría de los casos, una herencia simple es suficiente para representar esta relación. La generalización se describe con la frase “es un tipo de”.

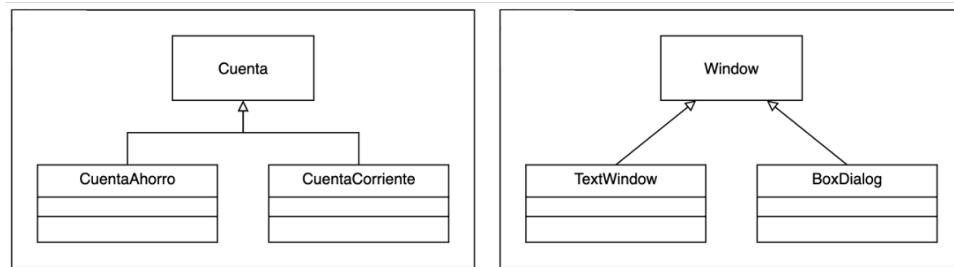


Figura 49: Generalización en clases

- **Asociación:** es una relación entre clases que indica que pueden existir vínculos entre los objetos de ambas. Así como los objetos son instancias de clases, los vínculos entre objetos son instancias de las asociaciones entre clases.

Las asociaciones pueden incluir:

- Nombre: describe la acción que el objeto origen realiza sobre el objeto destino.

- Nombres de roles: como alternativa al nombre, indica el “papel” que desempeña cada objeto en la relación.

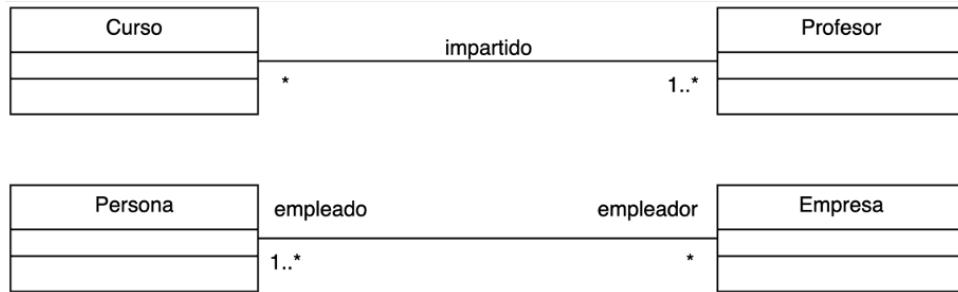


Figura 50: Asociación en clases

- Navegabilidad: permite restringir la dirección de la relación, señalando si una clase “conoce” a la otra.
- Multiplicidad: especifica cuántos objetos pueden estar involucrados en la relación en un momento dado, en uno o ambos extremos. Se define mediante un intervalo que indica el mínimo y máximo de objetos permitidos.

Existen tipos especiales de asociación:

- **Agregación:** es una relación entre una clase que representa un “todo” y otras clases que representan sus partes. En este caso, cada clase puede existir de forma independiente. Un ejemplo sería un ordenador y sus periféricos.
- **Composición:** es una forma más fuerte de agregación, en la que las partes y el “todo” están estrechamente ligados. A diferencia de la agregación, las partes en una composición no pueden existir independientemente del todo. Además, en la composición, cada parte pertenece exclusivamente a un único “todo”, mientras que en la agregación, las partes pueden ser compartidas entre varios elementos.

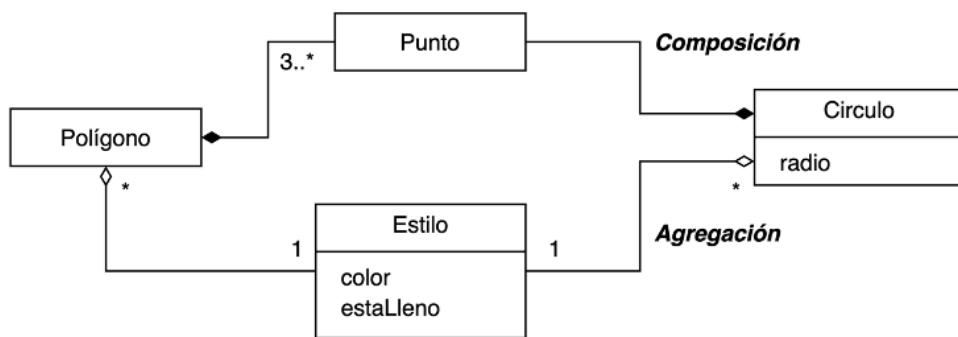


Figura 51: Agregación y Composición en clases

Se pueden crear **clases de asociación**, este tipo de clase es útil en relaciones de asociación con multiplicidad *, cuando existen atributos que no pertenecen a ninguna de las dos clases relacionadas. La clase de asociación forma parte de la relación, incluyendo todas sus propiedades.

Una clase de asociación actúa también como una clase independiente, pudiendo tener sus propios atributos y operaciones. Un ejemplo de su utilidad sería modelar que una persona puede tener diferentes contratos con una misma compañía a lo largo del tiempo, algo que no se podría representar adecuadamente solo con una asociación directa entre Persona y Compañía.

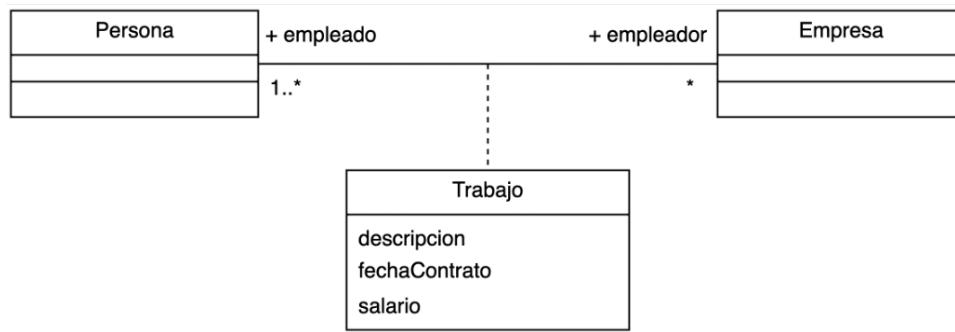


Figura 52: Clase de asociación

Los diagramas de clases son una herramienta central en el modelado orientado a objetos, ya que muestran un conjunto de clases, interfaces y colaboraciones, junto con sus relaciones. Estos diagramas parten de la idea de que el mundo real puede observarse a través de distintas abstracciones (enfoques subjetivos), empleando diversos mecanismos de abstracción, tales como:

- Clasificación / Instanciación
- Composición / Descomposición
- Agrupación / Individualización
- Especialización / Generalización

Entre estos, la clasificación es uno de los mecanismos de abstracción más utilizados en el modelado orientado a objetos.

En la Figura 53 se muestra un ejemplo de un diagrama de clases que modela el proceso de realización de un pedido, incluyendo una relación de generalización y varias asociaciones con distintas multiplicidades.

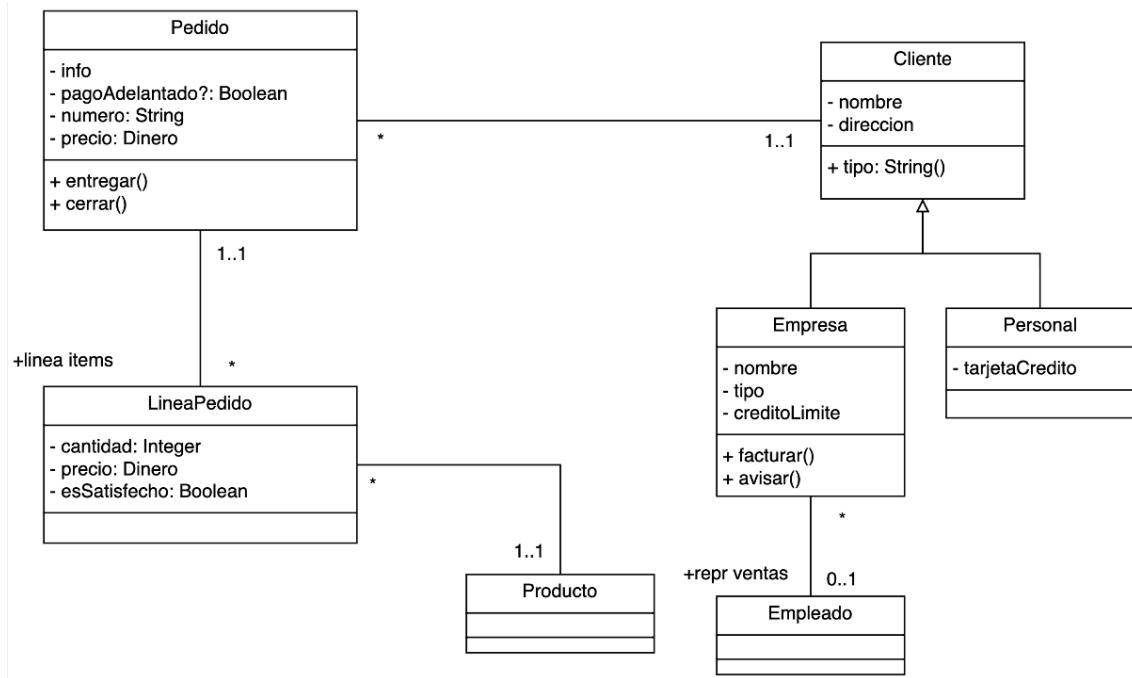


Figura 53: Ejemplo diagrama de clases

Perspectivas de los Diagramas de Clases

Existen diferentes perspectivas desde las cuales se pueden abordar estos diagramas, y cada una ofrece una visión específica del sistema en distintas etapas de su desarrollo. Las perspectivas definidas por Cook y Daniels [5] permiten modelar un sistema desde distintos niveles de abstracción, lo que facilita la comprensión y el diseño en función del contexto y los objetivos del proyecto. A continuación, se presentan las tres principales perspectivas de los diagramas de clases:

- **Conceptual**: Representa los conceptos del dominio, sin estar vinculada directamente a la implementación y es independiente del lenguaje de programación.
- **Especificación**: Define tipos de interfaces que pueden tener distintas implementaciones, según el entorno, el rendimiento o el proveedor.
- **Implementación**: Muestra las clases tal como se implementarán en el código. Es la perspectiva más común, aunque en algunos casos puede ser preferible usar la perspectiva de especificación.

Finalmente, en la Figura 54 se presenta otro diagrama de clases que modela el funcionamiento de una tienda y sus ventas.

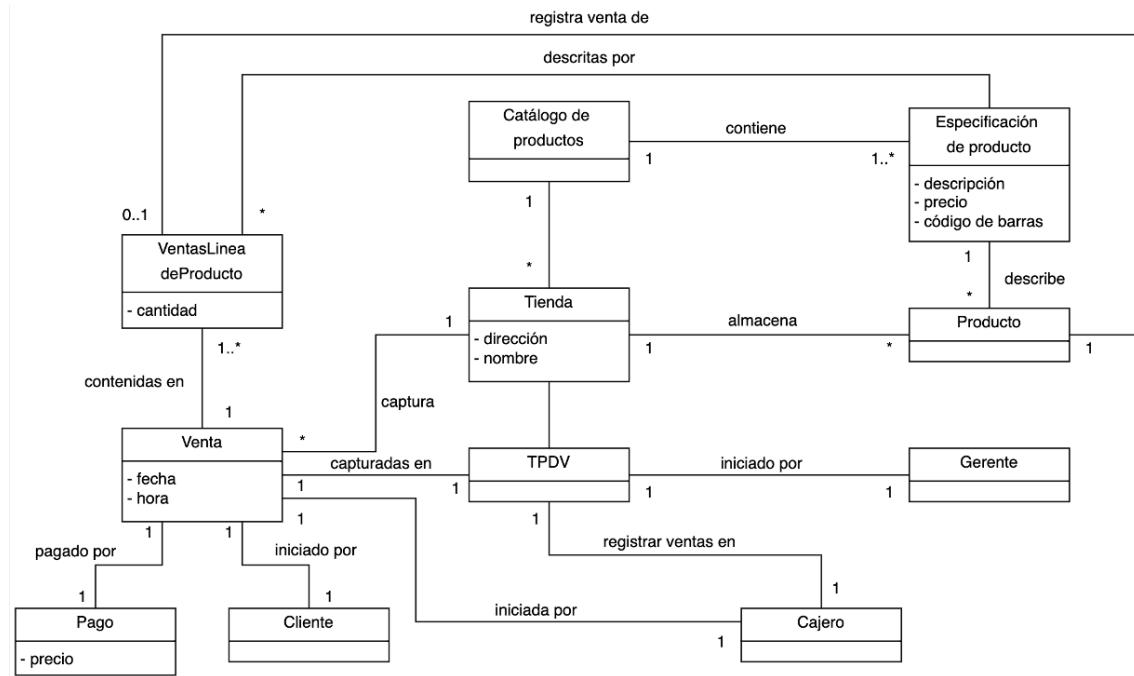


Figura 54: Ejemplo diagrama de clases

3.2. Diagrama de Objetos

Un **diagrama de objetos** proporciona una representación completa o parcial de los objetos en un instante específico de ejecución. Su principal objetivo es modelar las instancias de los elementos definidos en los diagramas de clases.

Tienen una serie de características:

- Representa una vista estática tanto del diseño como de los procesos del sistema.
- Cada instancia debe tener un nombre único dentro de su contexto.
- Las operaciones que se pueden ejecutar en el objeto se definen en su abstracción.
- Comparte la misma notación que los diagramas de clases, con la diferencia de que el nombre del objeto se subraya.

Los diagramas de objetos son útiles para:

- Ilustra las estructuras de datos y objetos dentro del sistema.
- Especifica detalles más precisos del modelo.
- Proporciona una “fotografía” del sistema en un momento específico de su ejecución.

Los objetos pueden desempeñar **roles** si están conectados mediante vínculos. Se pueden definir los siguientes vínculos:

- **Vínculo Bidireccional:** ambos elementos tienen conocimiento mutuo y pueden interactuar entre sí.

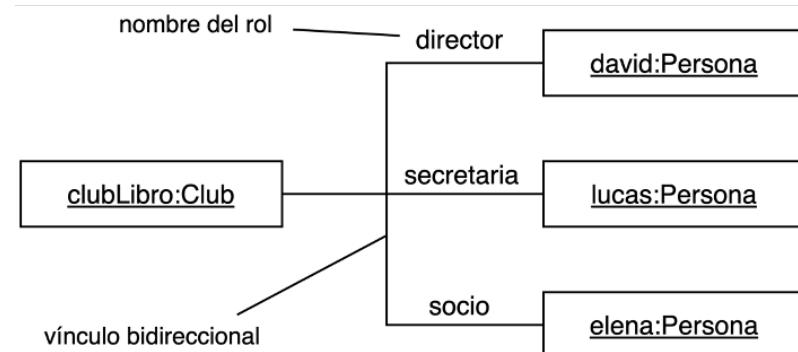


Figura 55: Vínculo bidireccional

- **Vínculo Unidireccional:** solo uno de los elementos tiene conocimiento del otro y puede interactuar con él, pero no al revés.

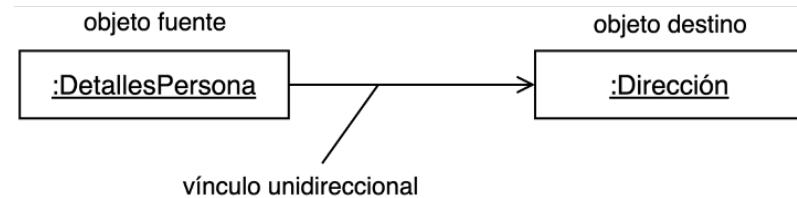


Figura 56: Vínculo unidireccional

Al visualizar el estado de un objeto, se muestra el valor de sus atributos en un momento específico, ya que el estado de un objeto es dinámico y puede cambiar a lo largo del tiempo.

Existen dos estereotipos en las relaciones de dependencia entre objetos y clases:

- ***instanceOf*:** indica que el objeto es una instancia de la clase especificada.
- ***instantiate*:** indica que la clase crea instancias de otra clase (Figura 57).

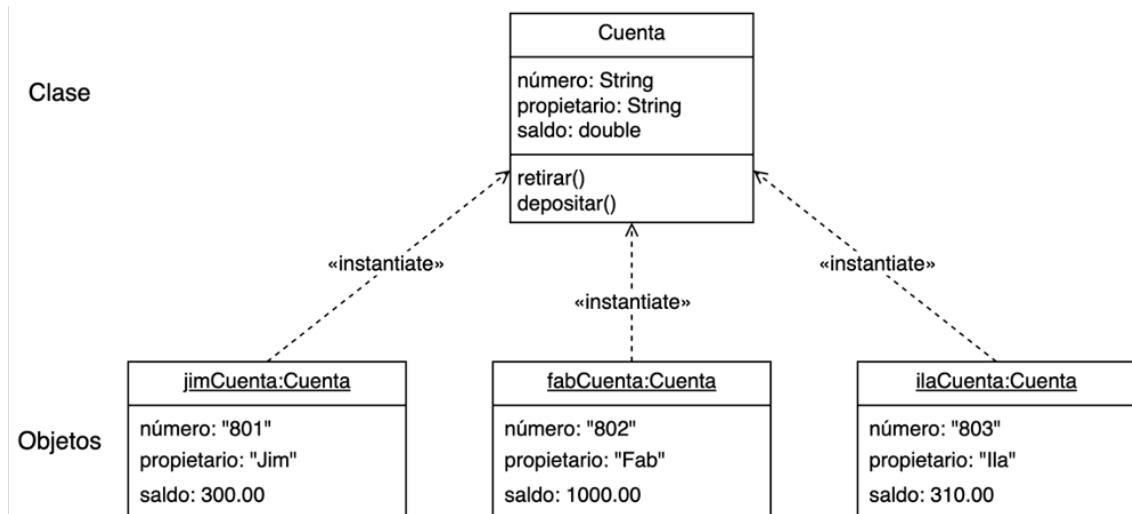


Figura 57: Objetos creados con *instantiate*

En la Figura 58 se presenta un ejemplo de un diagrama de objetos. En la parte superior se encuentra el diagrama de clases, y en la parte inferior se muestra el modelo de objetos generado a partir del diagrama de clases como ejemplo. Dado que existe una relación reflexiva, se puede representar la relación de un directorio dentro de otro.

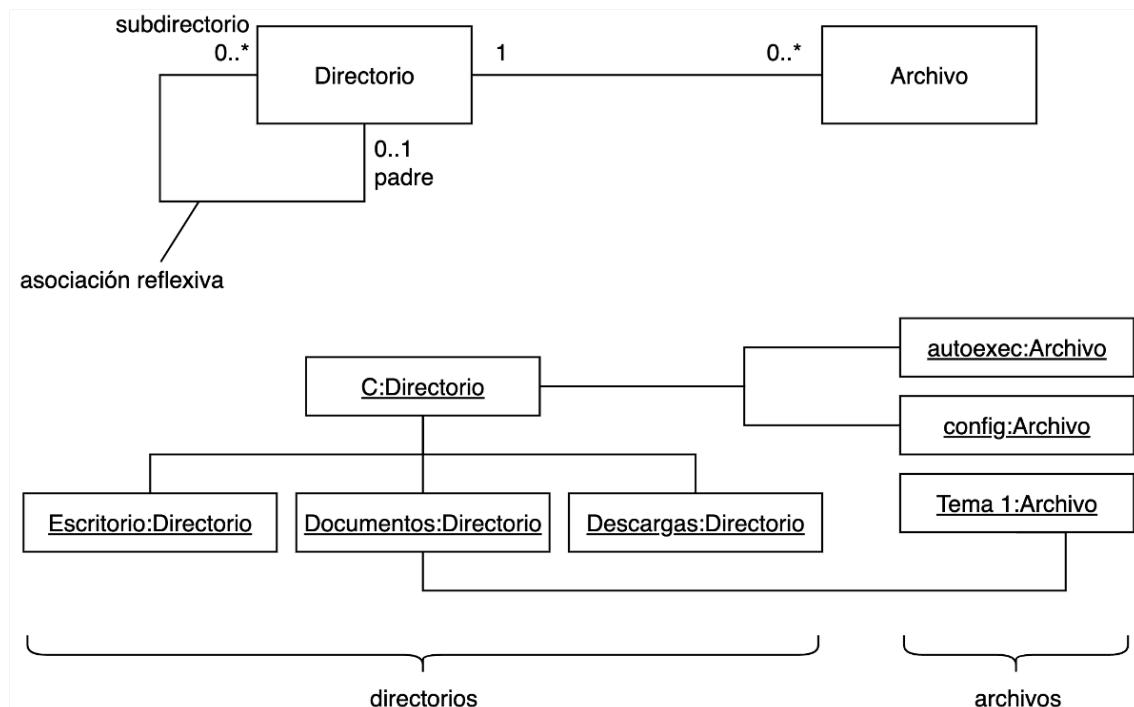


Figura 58: Ejemplo diagrama de objetos

3.3. Diagrama de Paquetes

Los **paquetes** se utilizan para organizar los elementos de modelado en unidades más grandes, que pueden ser manipuladas como un conjunto. Constituyen un mecanismo de agrupación dentro de UML, permitiendo organizar de manera estructurada los elementos del modelo.

Es importante controlar la visibilidad de los elementos dentro de un paquete, de modo que algunos sean accesibles desde fuera del paquete, mientras que otros permanezcan ocultos. Los elementos que están dentro de un mismo paquete suelen estar relacionados semánticamente y, por lo general, tienden a cambiar juntos. Por lo tanto, un paquete bien estructurado es cohesivo y tiene un bajo acoplamiento, con un control adecuado sobre el acceso a su contenido. Un paquete puede contener otros elementos como clases, interfaces, componentes, nodos, colaboraciones, casos de uso, diagramas y otros paquetes.

Cada elemento pertenece exclusivamente a un único paquete, y la visibilidad de los elementos contenidos en un paquete se controla de la misma manera que en las clases:

- **Público:** el elemento es visible para los contenidos de cualquier paquete que importe el paquete que lo contiene.
- **Privado:** el elemento no es visible fuera del paquete en el que se declara.

Los paquetes pueden estar relacionados mediante generalización y dependencias, estas últimas con semánticas propias como *merge* o *import*.

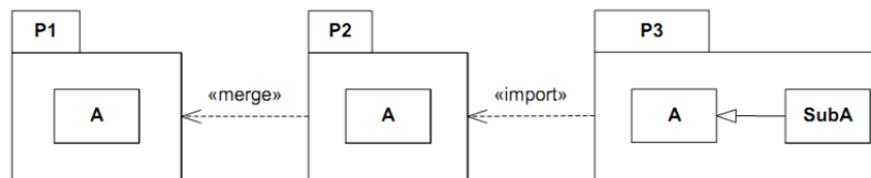


Figura 59: Dependencia entre paquetes

Gráficamente, un paquete se representa como una carpeta y debe tener un nombre que lo distinga de otros paquetes. Este nombre puede ser:

- **Nombre simple:** una cadena de texto única.
- **Nombre de camino:** el nombre del paquete precedido por el nombre del paquete contenedor.

Importación

La **importación** otorga un permiso unidireccional que permite a los elementos de un paquete acceder a los elementos de otro paquete. En UML, la relación de importación se modela como una dependencia con el estereotipo *import*.

Las partes públicas de un paquete se denominan **exportaciones**. Los elementos que un paquete exporta solo son visibles para aquellos paquetes que lo importan explícitamente.

Las dependencias entre paquetes **no son transitivas**, lo que facilita la implementación de una arquitectura por capas. Esto significa que si un elemento es visible dentro de un paquete, será visible también en todos los paquetes que estén incluidos dentro de este paquete.

Los paquetes anidados pueden acceder a todos los elementos de los paquetes que los contienen.

En la Figura 60 se muestra ejemplos de importaciones y exportaciones entre paquetes.

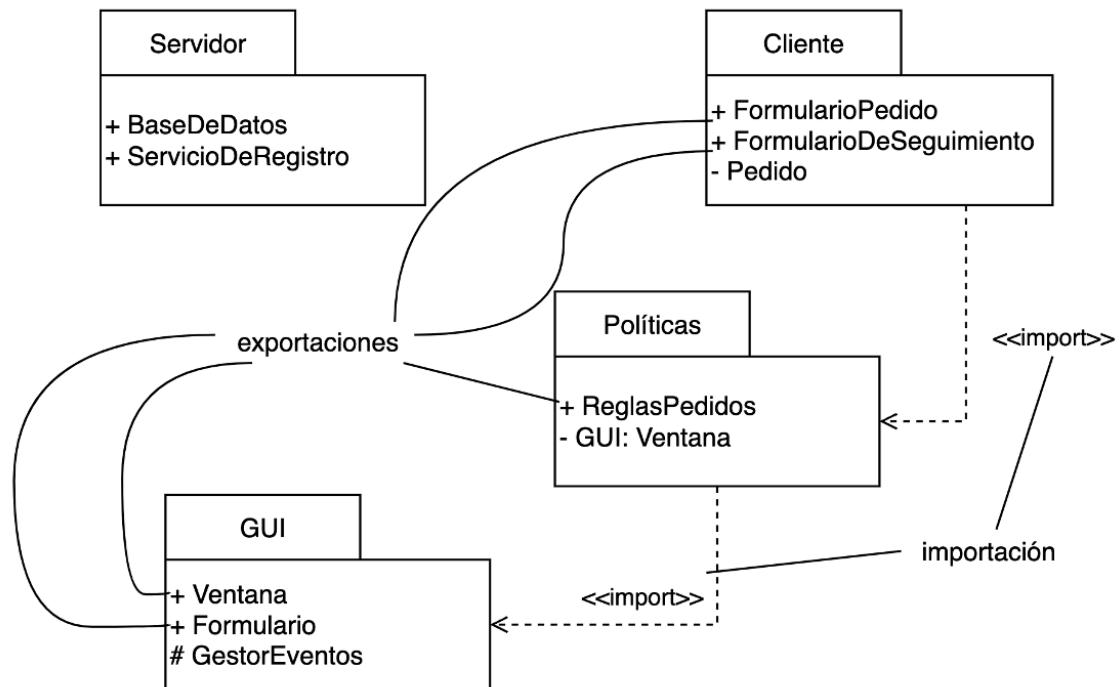


Figura 60: Ejemplo de importación entre paquetes

Generalización

La generalización es otro tipo de relación que puede existir entre paquetes y se utiliza para definir familias de paquetes. Los paquetes involucrados en esta relación siguen el mismo principio de sustitución que se aplica a las clases. En la Figura 61 se puede observar un ejemplo y como un paquete hereda atributos de otro.

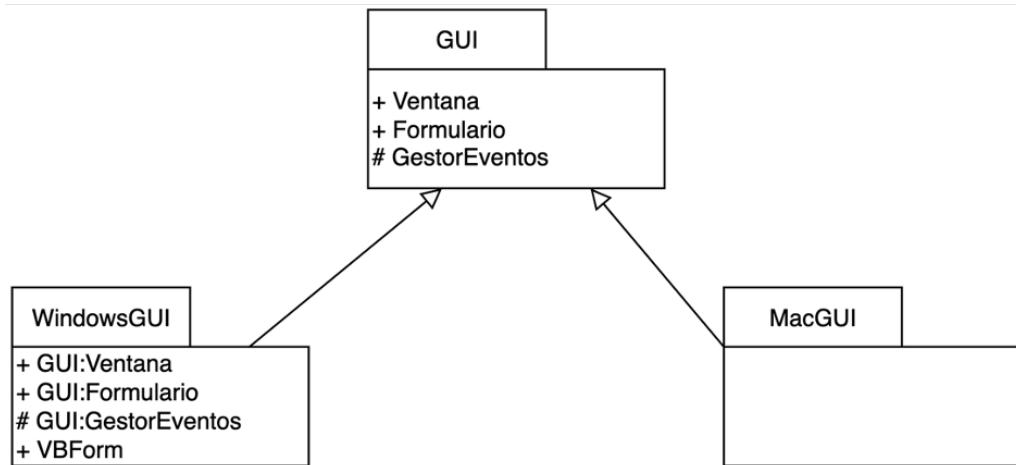


Figura 61: Ejemplo de generalización entre paquetes

3.4. Diagrama de Componentes

El diagrama de componentes muestra cómo un sistema se organiza en **componentes** y las relaciones entre ellos. Este tipo de diagrama tiene un nivel de abstracción superior al de los diagramas de clases, ya que un componente suele estar implementado por una o más clases en tiempo de ejecución.

Principalmente, se emplea en el ámbito de la arquitectura de software y resulta útil para:

- Modelar la vista lógica de un sistema.
- Representar el código fuente.
- Mostrar las diferentes versiones ejecutables.
- Modelar bases de datos físicas.
- Representar sistemas adaptables.

Un componente es una unidad autónoma dentro de un sistema, definida por sus interfaces. Su propósito es encapsular una parte de las funcionalidades del sistema, accesibles únicamente a través de sus interfaces. Un componente puede ser reemplazado por otro siempre que cumpla con la misma especificación, ya que es autocontenido y reemplazable.

La estructura interna de un componente puede modelarse para delegar la realización de sus interfaces a alguna de sus partes internas. Los componentes también pueden depender de otros, lo que en realidad significa que dependen de las interfaces de esos otros componentes. Además, los componentes pueden tener estereotipos para añadir información semántica, como «subsystem».

Existen varios tipos de componentes:

- Ejecutables: Se ejecutan de forma autónoma.
- Librerías: Biblioteca de objetos estática o dinámica.
- Tabla: Representa una tabla en una base de datos.
- Archivo: Contiene código fuente o datos.
- Documento: Representa otro tipo de documento.

El otro elemento esencial del modelo arquitectónico son las interfaces. Estas especifican un conjunto de características públicas (habitualmente servicios) que

otros elementos pueden utilizar. Su propósito principal es separar la especificación (qué hace) de la implementación (cómo lo hace).

Una interfaz puede tener múltiples implementaciones válidas, siempre que cumplan las condiciones establecidas en la propia interfaz. Los atributos y operaciones de una interfaz deben estar completamente especificados, actuando como un “contrato”:

- Firma completa de la operación: nombre, parámetros y valor de retorno.
- Semántica de la operación: puede expresarse en texto o pseudocódigo.
- Nombre y tipo de atributos.
- Restricciones de uso o comportamiento.

En los diagramas de componentes se representan las siguientes relaciones, cada una con una función específica en la arquitectura del sistema:

- **Dependencia:** Indica que un componente depende de otro para su funcionamiento. Esto significa que un cambio en el componente del cual depende podría afectar al componente dependiente, ya sea en la implementación o en el comportamiento.



Figura 62: Relación de dependencia

- **Generalización:** Relación en la cual un componente se considera una especialización de otro. Permite definir componentes genéricos y extender sus funcionalidades en componentes específicos que heredan sus características.



Figura 63: Relación de generalización

- **Proporciona (Interfaz):** Esta relación muestra que un componente ofrece una interfaz que otros componentes pueden utilizar. La interfaz proporcionada define el conjunto de operaciones o servicios que el componente pone a disposición de otros elementos del sistema. Una interfaz proporcionada se representa como un pequeño círculo unido a uno de los lados del componente. El círculo está conectado al componente que proporciona la interfaz y sirve para indicar las operaciones que el componente pone a disposición de otros.



Figura 64: Relación que proporciona

- **Consumo (Interfaz):** Representa que un componente utiliza una interfaz proporcionada por otro componente. Este tipo de relación es fundamental para mostrar la interacción entre componentes, donde uno depende de la funcionalidad especificada en la interfaz de otro. Una interfaz consumida se representa mediante una línea que termina en un semicírculo o “socket” que se conecta a la interfaz proporcionada por otro componente. Este semicírculo indica que el componente necesita o consume los servicios definidos por la interfaz del componente al que está conectado.



Figura 65: Relación que consume

Este diagrama muestra cómo los componentes se interrelacionan, a través de sus interfaces, para conformar la arquitectura completa de un sistema. Puede incluir, o no, la composición interna de los componentes o la especificación detallada de sus interfaces. A través de este tipo de diagramas, es posible representar patrones arquitectónicos, como la arquitectura en capas. En la Figura 66 se ilustran dos ejemplos.

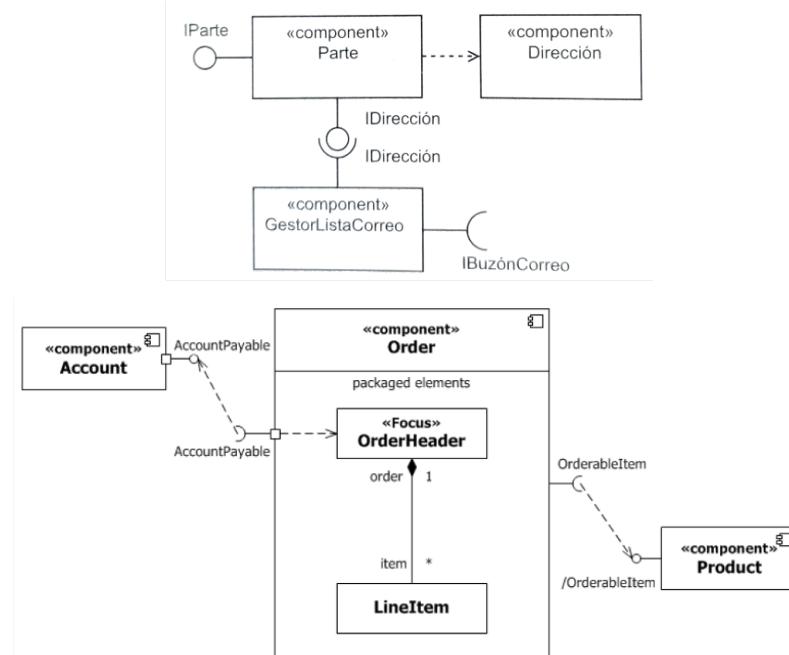


Figura 66: Ejemplos diagrama de componentes

3.5. Diagrama de Despliegue

El diagrama de despliegue representa la topología de hardware de un sistema. Este diagrama captura la relación entre los elementos de un modelo conceptual o físico y los recursos de información asignados a ellos. Se utiliza principalmente en el ámbito de la arquitectura, siendo desarrollado por diseñadores, ingenieros de sistemas e ingenieros de redes.

Es útil para:

- Indicar la distribución de los componentes.
- Evaluar el rendimiento y la carga del hardware del sistema.
- Analizar aspectos como redundancia, balance de carga, entre otros.

Los **nodos** son elementos físicos en tiempo de ejecución que representan recursos computacionales, como procesadores o dispositivos. Un nodo puede contener objetos o instancias y se utiliza para modelar la topología de hardware donde se ejecuta el sistema. Cada nodo debe contar con un nombre único, que puede ser simple o incluir una ruta de paquete. Los nodos también pueden incorporar valores etiquetados o compartimentos adicionales para mostrar información adicional. Los nodos se representan como una caja (Figura 67).

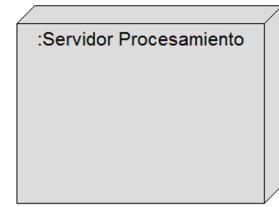


Figura 67: Nodo

Las **relaciones** conectan los distintos componentes dentro del diagrama de despliegue. En una relación, es posible representar:

- El tipo de comunicación entre componentes, utilizando una etiqueta.
- La cardinalidad de la relación.

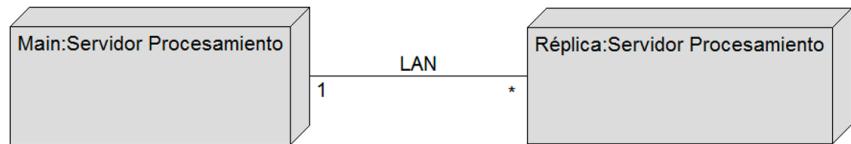
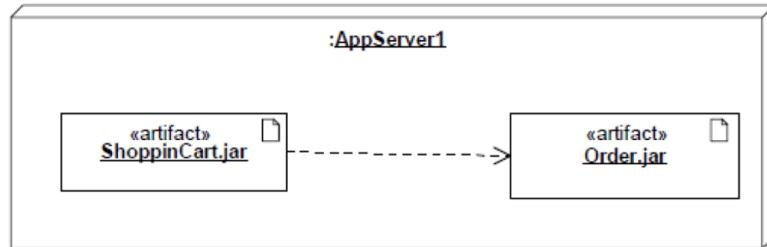


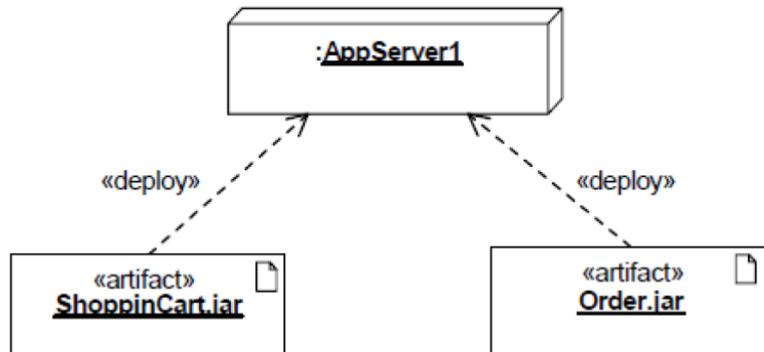
Figura 68: Relaciones en diagramas de despliegue

Los **artefactos** representan elementos de implementación en un sistema. Pueden ubicarse de dos maneras:

- **Dentro de los nodos:** especificando el recurso computacional en el que se ejecutarán (Figura 69a).
- **Mediante relaciones:** sin indicar un recurso de ejecución específico (Figura 69b).



(a) Artefactos dentro de un nodo



(b) Artefactos como relaciones

Figura 69: Ubicación de los artefactos

Los artefactos son los elementos que se despliegan en los nodos y pueden asociarse con archivos fuente, ejecutables, librerías, scripts, tablas de base de datos y documentos.

Estereotipos:

- Se emplea «artifact» para denotar un artefacto general
- También pueden especificarse estereotipos más detallados como «script», «executable», «library», «document», entre otros.

En la Figura 70 se muestra un ejemplo de diagrama de despliegue.

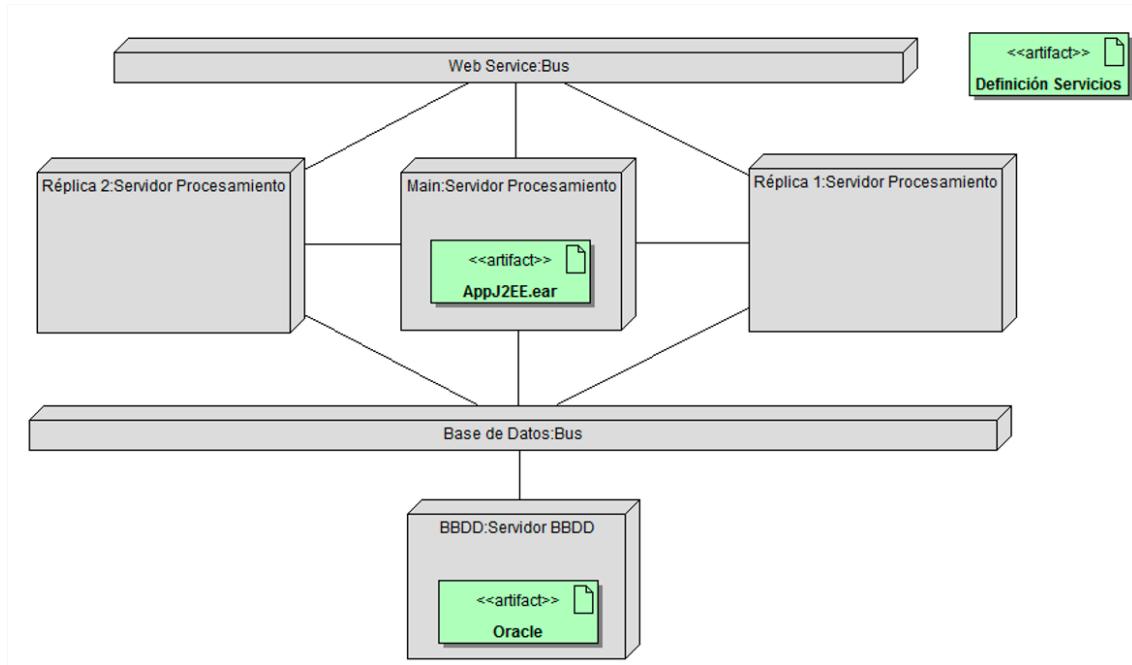


Figura 70: Ejemplo diagrama de despliegue

4. Técnicas Estructuradas

Según Yourdon no existe un criterio definitivo por el que podamos organizar las técnicas de la metodología estructurada. Aunque si podemos considerar dos enfoques:

- Según el enfoque de **representación**, clasifica las técnicas según la forma en que se representan los componentes y aspectos del sistema.
- Según el enfoque de **modelado**, clasifica las técnicas en función de los modelos del sistema que se crean.

4.1. Clasificación según el Enfoque de Representación

El enfoque de representación organiza las técnicas de acuerdo con la forma en que presentan o describen los elementos y componentes del sistema, permitiendo diferentes niveles de detalle y especificación.

- **Gráficas:** Utilizan iconos o símbolos para representar visualmente los componentes de un aspecto específico del modelo. Cada técnica gráfica puede enfocarse en diferentes aspectos del sistema, como funciones o flujos de datos, y suelen combinarse con otros tipos de técnicas para ofrecer una visión integral del sistema.
- **Textuales:** Ofrecen una descripción detallada de los componentes definidos en los diagramas gráficos, usando una gramática formal o semiformal. Las representaciones textuales suelen complementarse con representaciones gráficas, proporcionando mayor especificidad y exactitud en la descripción de los componentes.
- **Marcos o Plantillas:** Sirven para organizar y especificar información relacionada con componentes de un modelo ya declarados en diagramas o esquemas. Se representan mediante formularios y se utilizan para documentar las propiedades y características de los componentes de forma estructurada.
- **Matriciales:** Estas técnicas no se utilizan para definir componentes del sistema, sino para verificar la precisión y la completitud de los modelos. Permiten estudiar las referencias cruzadas entre los componentes, ayudando en la validación y en el aseguramiento de la consistencia entre diferentes modelos.

4.2. Clasificación según el Enfoque de Modelado

El enfoque de modelado organiza las técnicas según el tipo de modelos del sistema que se crean, distinguiendo entre la funcionalidad, los datos, y los eventos temporales. A continuación se detallan las técnicas específicas según cada dimensión:

1. Modelado basado en la Información

La dimensión de la información se centra en representar las **entidades** del sistema y sus **relaciones**. Se enfoca en describir los datos que el sistema maneja, cómo se transforman durante el procesamiento y cómo se generan como salida.

Su objetivo principal es capturar el dominio de datos del sistema, especificando cada ítem de datos que se acepta, procesa y produce como resultado.

Las principales técnicas que se utilizan son:

- **Diagramas Entidad-Relación (ER):** Representan las entidades en el sistema y las relaciones entre ellas.
- **Diagramas de Estructura de Datos:** Detallan la organización interna de los datos del sistema.
- **Matriz Entidad/Entidad:** Ayuda a identificar y organizar las relaciones entre distintas entidades.

2. Modelado basado en el Tiempo

La dimensión del tiempo analiza cómo responde el sistema a los **eventos** que ocurren en momentos específicos. Esta dimensión se enfoca en describir las respuestas temporales del sistema, cuando ciertos eventos activan sus funciones o procesos.

Su objetivo es describir el momento en el cual se activa una función o se ejecuta un proceso en respuesta a un evento, gestionando así la secuencia de acciones en el sistema.

Las principales técnicas que se utilizan son:

- **Diagramas de Transición de Estados:** Representan los posibles estados del sistema y las transiciones entre ellos en respuesta a eventos.
- **Listas de Eventos:** Enumeran y describen los eventos que desencadenan alguna respuesta en el sistema.

- **Redes de Petri:** Modelan secuencias y paralelismos de eventos en el sistema.
- **Diagramas de Historia de la Vida de las Entidades:** Describen el ciclo de vida de cada entidad en función de los eventos que afectan su estado.
- **Matriz Evento/Entidad:** Relaciona los eventos con las entidades que afectan o dependen de estos eventos.

3. Modelado basado en la Función

La dimensión de la función se enfoca en representar las **funciones y procesos** del sistema y las interfaces que operan sobre los datos. Este enfoque estudia cómo el sistema realiza sus tareas y ejecuta transformaciones sobre los datos.

Tiene como objetivo especificar las funciones, procesos o transformaciones del sistema, detallando cómo opera y qué tipo de procesamiento realiza sobre la información que maneja.

Sus técnicas principales son:

- **Diagramas de Flujo de Datos (DFD):** Representan las funciones del sistema, sus entradas y salidas, y las interfaces entre ellas.
- **Lenguaje Estructurado:** Se utiliza para definir procesos o funciones con mayor detalle, permitiendo una especificación formal o semiformal.
- **Árboles de Decisión:** Representan decisiones complejas en forma jerárquica, mostrando las alternativas posibles.
- **Tablas de Decisión:** Organizan las reglas de decisión en una tabla, mostrando las acciones a tomar en función de condiciones específicas.
- **Diagramas de Descomposición Funcional:** Descomponen el sistema en sus subfunciones o módulos principales.
- **Matriz Función/Entidad:** Relaciona las funciones con las entidades que participan en cada una.
- **Diccionario de Datos:** Define y documenta los elementos de datos, proporcionando un glosario de términos y especificaciones para cada componente de datos en el sistema.

4.2.1. Diagramas de Flujo de Datos

El Diagrama de Flujo de Datos (DFD) es una de las técnicas más utilizadas en el análisis estructurado, concebida para ayudar a los analistas a modelar tanto las funciones que debe realizar el sistema como los datos que fluyen entre ellas. Surge a finales de la década de 1970, cuando Tom De Marco formaliza este enfoque. Su propuesta incluía herramientas de soporte como el Diccionario de Datos (DD) y las Especificaciones de Procesos, que complementan el DFD y permiten una descripción detallada y estructurada de los componentes del sistema.

Un DFD es una representación gráfica en forma de red que ilustra el flujo de información y las transformaciones que esta experimenta al moverse desde la entrada hasta la salida del sistema. Esta técnica permite modelar las funciones y datos del sistema en distintos niveles de abstracción, proporcionando una visión clara y estructurada del sistema en su totalidad.

Para lograr este objetivo, el sistema se modela mediante un conjunto de DFD nivelados. Los niveles superiores describen las funciones generales del sistema, mientras que los niveles inferiores desglosan estas funciones en tareas más detalladas, ofreciendo un análisis progresivamente más específico.

Los componentes principales de un DFD son: Procesos, Almacenes, Entidades Externas y Flujo de Datos (Figura 71). Cada uno de estos componentes cumple un rol específico en el modelado funcional del sistema, facilitando la comprensión y comunicación de cómo operan las funciones y cómo se gestionan los datos en la solución de software.

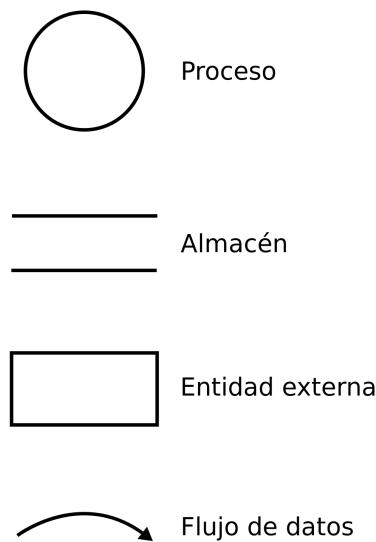


Figura 71: Representación gráfica de los componentes de un DFD

Procesos

Los procesos representan funciones que transforman los flujos de datos de entrada en uno o más flujos de salida, y se refieren a operaciones que el sistema debe realizar, no a programas en ejecución.

Cada proceso debe cumplir con la regla de conservación de los datos, es decir, generar sus salidas a partir de las entradas y cualquier información adicional. Si faltan datos necesarios para producir las salidas, se produce un error de conservación de datos; si algunos datos de entrada no se utilizan, se considera una pérdida de información.

Para identificar los procesos en los DFD, existen algunas normas de nomenclatura:

- Numeración y nombre únicos en el conjunto de DFD.
- Nombre breve y preciso que refleje la función completa del proceso.
- Independencia de la implementación física en los DFD lógicos, que representan solo la lógica del sistema, sin detalles técnicos de realización.

Almacenes de Datos

Los almacenes de datos representan información almacenada temporalmente en el sistema, conocida como datos “en reposo”. Funcionan como dispositivos lógicos de almacenamiento, y pueden contener cualquier tipo de datos, sin importar el tipo de tecnología o dispositivo físico utilizado.

La representación gráfica de un almacén de datos varía según la metodología utilizada. Todos los almacenes de datos deben llevar un nombre que describa de forma representativa el tipo de información que contienen.

Tienen una serie de características:

- **Legibilidad:** Un almacén puede repetirse en un DFD para mejorar su claridad.
- **Ubicación en Niveles:** En DFD nivelados, el almacén debe aparecer en el nivel más alto en que interconecta múltiples procesos y ser representado en niveles inferiores solo donde interactúe directamente con procesos específicos.
- **Almacenes Locales:** Si un almacén se conecta solo con un proceso en un nivel, debe considerarse “local” y representarse en el nivel donde se detalla dicho proceso.

Existen dos tipos de estructuras de almacén:

- **Estructura Simple:** Similar a un registro, compuesto de atributos donde uno o más identifican una ocurrencia.
- **Estructura Compleja:** Representada idealmente mediante un Diagrama Entidad-Relación (ER) para describir su organización interna.

La información de cada almacén debe documentarse en el Diccionario de Datos para asegurar precisión y consistencia en los diagramas y la documentación del sistema.

Entidades Externas

Las entidades externas representan elementos que generan o consumen información del sistema sin pertenecer a él. Pueden corresponder a un subsistema, una persona, un departamento, una organización o cualquier otra fuente que proporcione datos al sistema o reciba información de él. Su representación gráfica puede variar según la metodología empleada.

Dado que las entidades externas no forman parte del sistema, los flujos de información que emiten o reciben establecen la interfaz entre el sistema y su entorno. Para facilitar su identificación, cada entidad debe tener un nombre claro y representativo que describa su relación o función respecto al sistema. Si mejora la claridad, una entidad externa puede aparecer varias veces en un mismo DFD.

Generalmente, estas entidades se encuentran en el Diagrama de Contexto, el DFD de nivel más alto, donde se delimitan los límites y la interacción del sistema con el entorno. No obstante, en algunos casos, pueden incluirse en otros diagramas de niveles inferiores para mejorar la comprensión del sistema.

Flujo de Datos

Un **flujo de datos** es un camino por el que circulan datos de composición definida entre distintas partes del sistema, representada mediante arcos dirigidos. Según su persistencia, los flujos de datos se dividen en:

- **Flujos de Datos Discretos:** Son aquellos que trasladan información en un momento específico, como una solicitud de datos realizada en un instante concreto.
- **Flujos de Datos Continuos:** Un caso especial de flujo discreto, donde los datos permanecen activos a lo largo del tiempo, como en un proceso que mo-

nitorea continuamente el estado de un sistema o la disponibilidad de un dispositivo.

En la Tabla 5 se observa que existen dos tipos de conexiones que relacionan las entidades externas y los almacenes (*) que indica que solo en el caso en que el almacén de datos externo sirva de interfaz entre el sistema y la entidad externa será permitido este tipo de conexión y solo aparecerá en el diagrama de contexto.

Destino/Fuente	Proceso	Almacén	Entidad Externa
Proceso	SI	SI	SI
Almacén	SI	NO	NO*
Entidad Externa	SI	NO*	NO

Tabla 5: Conexiones entre elementos de un DFD

La conexión directa entre dos procesos mediante un flujo de datos es posible si la información es síncrona, es decir, si el proceso receptor comienza su actividad justo cuando el proceso emisor ha completado la suya (Figura 72).

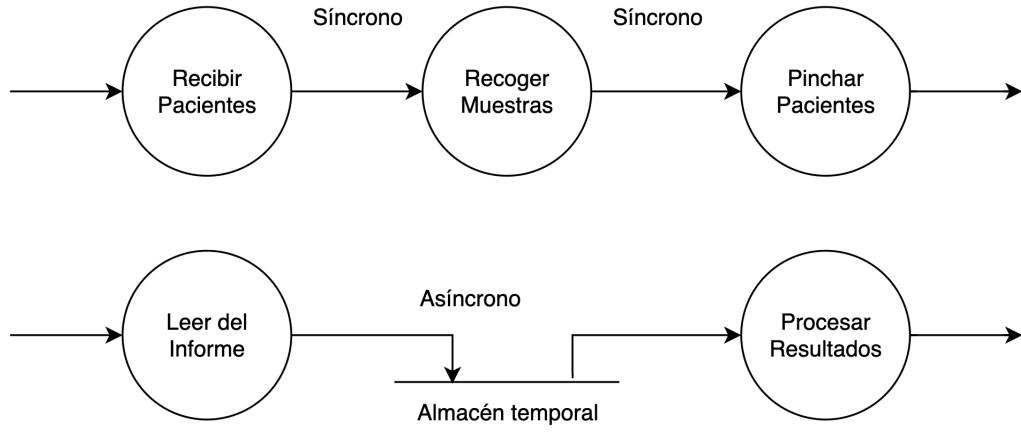


Figura 72: Conexión sincrónica en DFD

Las diferentes conexiones que se pueden hacer entre los procesos y almacenes son:

- **Flujo de Consulta:** Permite que el proceso acceda al almacén para consultar atributos específicos o verificar si ciertos valores cumplen criterios predefinidos.
- **Flujo de Actualización:** Implica la modificación del almacén mediante la creación, eliminación, o actualización de valores de entidades o relaciones existentes.

- **Flujo de Diálogo:** Combina consulta y actualización en un flujo interrelacionado, permitiendo tanto acceder a información como modificarla en un solo proceso.

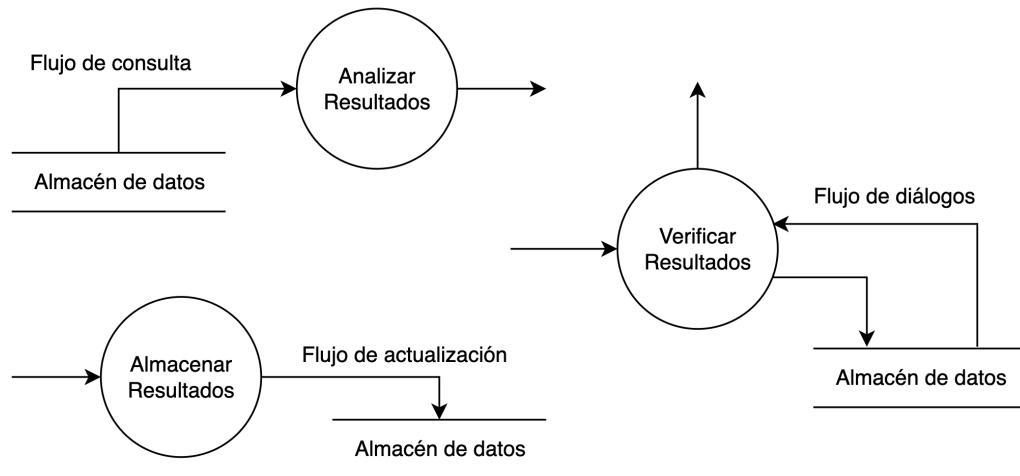


Figura 73: Conexiones entre procesos y almacenes de un DFD

Cada flujo de datos debe tener un nombre representativo que refleje claramente el tipo de información que transporta. Sin embargo, no es necesario nombrar los flujos de datos que ingresan o salen de almacenes de estructura simple, ya que en estos casos su estructura corresponde a la del almacén.

Es importante destacar que los flujos de datos no indican el control de la ejecución de los procesos, ni definen cuándo debe comenzar o finalizar un proceso.

Niveles de Abstracción en los DFD

En los DFD para sistemas grandes, se recomienda estructurarlos en niveles de abstracción mediante una aproximación descendente (top-down), donde cada nivel ofrece una vista más detallada de las funciones representadas en el nivel anterior.

Un DFD de un sistema grande se compone de un conjunto de DFD organizados jerárquicamente, donde cada nivel se expande sobre el anterior.

- **Diagrama de Contexto (o Diagrama de Nivel 0):** es el más general, definiendo los límites del sistema e identificando las interfaces de entrada y salida con el entorno.
- **Diagrama de Niveles:** detallan las funciones principales del sistema y sus relaciones, y cada función de un nivel puede descomponerse en niveles inferiores

para mostrar más detalle.

- **Procesos Primitivos:** representan las funciones básicas que no se descomponen más, describiéndose con precisión, ya que son los bloques fundamentales del sistema.

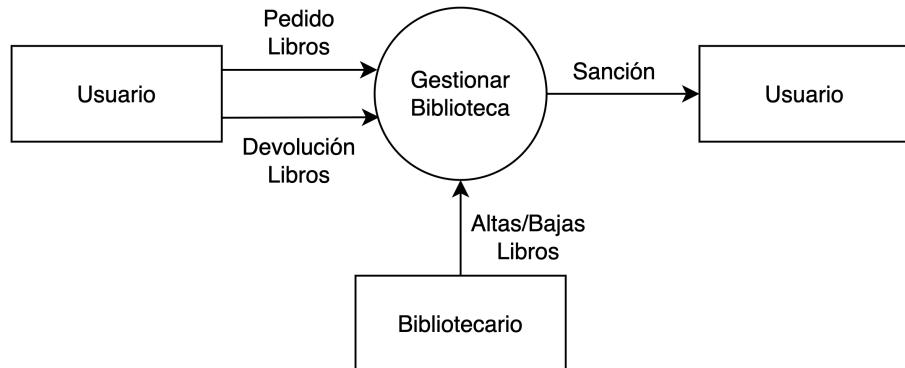
Los niveles de descomposición suelen organizarse de la siguiente manera:

- **Nivel 0:** Diagrama de Contexto, que define los límites generales del sistema.
- **Nivel 1:** Diagrama de Subsistemas, que representa las principales divisiones funcionales del sistema.
- **Nivel 2:** Diagramas de las Funciones de los Subsistemas, detallando las principales operaciones dentro de cada subsistema.
- ...
- **Nivel N:** Diagramas de Procesos, detallando cada subfunción hasta el nivel requerido.

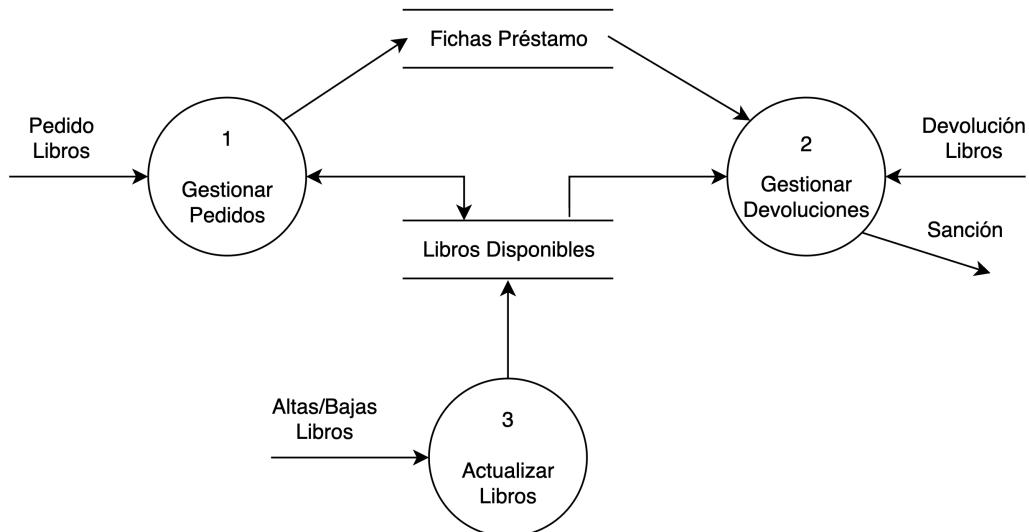
Para asegurar consistencia, los flujos de datos en cada nivel deben coincidir con los flujos de datos en el nivel inferior según la **regla de balanceo**, que exige que todos los flujos de datos que entran en un diagrama hijo estén representados en el diagrama padre. Las salidas también deben coincidir, salvo en casos de rechazos triviales.

La numeración de los DFD sigue un esquema estructurado: el Diagrama de Contexto se numera como 0, los Diagramas de Nivel 1 se numeran del 1 al N, y los niveles sucesivos se numeran según un sistema de numeración anidada o sistema Dewey, facilitando la referencia y coherencia en la documentación.

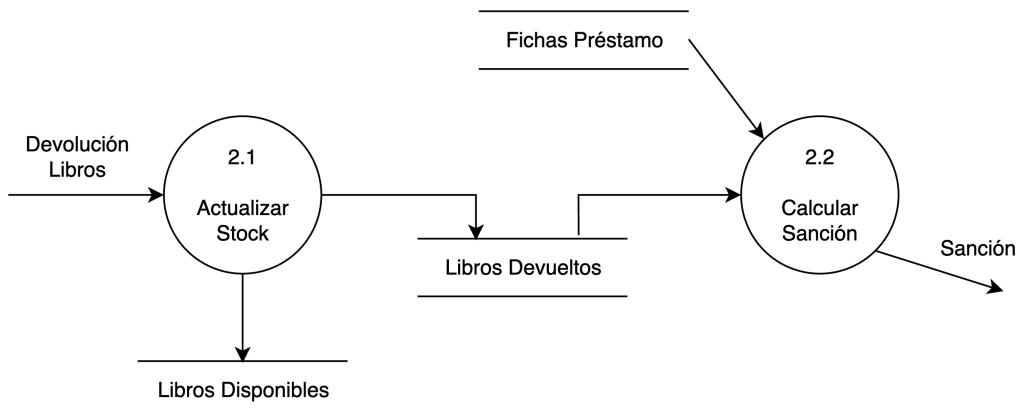
En la Figura 74a se muestra un diagrama de contexto de la gestión de una biblioteca, en la Figura 74b el diagrama de nivel 1 y finalmente en la Figura 74c el diagrama de las funciones de los subsistemas.



(a) Diagrama de Contexto



(b) Diagrama 1: Gestionar Biblioteca



(c) Diagrama 2: Gestionar Devoluciones

Figura 74: Ejemplo de diagrama de flujo de control

Referencias

- [1] Stachowiak H, Allgemeine modelltheorie, Springer, 1973.
- [2] Irene T. Luque Ruiz, Apuntes de la Asignatura Ingeniería del Software (2013).
- [3] G. Booch, J. Rumbaugh y I. Jacobson, El Lenguaje Unificado de Modelado, Pearson Education, 2006.
- [4] J. Arlow y I. Neustadt, UML 2, Anaya, 2005.
- [5] Cook, S. y Daniels, J., Designing Objects Systems, Prentice Hall, 1994.