UML
Guía Visual

Cómo crear
formas de vida
organizativa

info@vico.org

Rev. 0.17

Josep Vilalta

UML
Guía Visual

Cómo crear formas de vida organizativa

Presentación

Esta guía describe como definir, organizar y visualizar lo que denominamos formas de vida organizativa (VIO) con la notación Unified Modelling Language (UML). Una VIO representa un ciclo de actividad realizado por uno o varios agentes con un propósito concreto, en base a una práctica consensuada para utilizar los recursos disponibles y para tomar las decisiones que caracterizan el comportamiento de una organización.

A diferencia de los sistemas biológicos, las VIO nacen y se desarrollan a partir de una voluntad compartida, de una idea y de un liderazgo. Pero de la misma manera que la selección natural actúa en los sistemas biológicos, la continuidad de una VIO está condicionada a la implementación eficiente de sus procesos esenciales. Conocer estos procesos, saber como aplicar los recursos y como tomar las decisiones para satisacer la cadena de valor de todos los agentes son los factores que toda organización ha de tener en cuenta para evolucionar y asegurar su viabilidad.

La notación UML (no hay que confundir con las metodologías que utilizan dicha notación), se ha convertido desde finales de los 90 en un estándar para modelar con tecnología orientada a objetos todos aquellos elementos que configuran la arquitectura de un sistema de información y, por extensión, de los procesos de negocio de una organización. De la misma manera que los planos de un arquitecto disponen el esquema director a partir del cual levantamos un edificio, los diagramas UML suministran un modelo de referencia para formalizar los procesos, reglas de negocio, objetos y componentes de una organización. La interacción de todos estos elementos es una representación de nuestra realidad.

Nuestros límites para entender esta realidad están en nuestro lenguaje. El mundo es la suma total de lo que podemos definir, organizar y visualizar. Cabe preguntarse ¿de qué manera un modelo en UML representa nuestra experiencia?. Enseñar a utilizar un lenguaje formal siempre es problemático. Es necesario *mostrar* como este lenguaje puede ser aplicado a la realidad tal como la conocemos y tal como la compartimos con los demás. Con esta guía visual mostramos de manera precisa las técnicas básicas para usar UML en diferentes contextos. La clave está en discriminar cuales son aquellos procedimientos esenciales que nos permiten evitar costosas confusiones conceptuales. No es mediante el descubrimiento de nuevos objetos y de sus múltiples conexiones que avanzamos en las respuestas a nuestros interrogantes cuando modelamos un dominio, sino mediante la disolución de las contradicciones que existen entre los términos (conceptos) ya conocidos y, en último caso, mediante la reducción de su número despejando aquellos conceptos que no añaden valor a la comprensión de dicho dominio. Reconsiderar lo obvio, desenmascarar presunciones, desambigüar conceptos conocidos, todo en busca de la *simplicidad* y la *usabilidad*.

La tecnología orientada a objetos persigue el antiguo principio del divide y vencerás. Su objetivo es descomponer la *complejidad* en partes más manejables y comprensibles. No parece que esto sea algo novedoso con respecto a la tradicional descomposición funcional de los métodos estructurados. Sin embargo, la gran diferencia reside en aplicar la dualidad *estructura-función* en pequeñas unidades capaces de comunicarse y reaccionar en base a la aparición de una serie de eventos. El esquema dominante de la separación de estructuras de datos y funciones (bases de datos y programas) está amenazado pero aún se resiste a desaparecer.

Mucha gente cree que la principal utilidad de la orientación a objetos es la *reutilización del código* para conseguir un desarrollo más rápido de las aplicaciones (Rapid Application Development). No comparto esta opinion. Si hay algo que caracteriza un entorno de desarrollo actual es la constante del *cambio*. Todo proyecto que sobrepase los tres meses está amenazado por la aparición de nuevas plataformas más exigentes, la extinción de herramientas sin previo aviso y, de manera sistemática, por la rotación del personal crítico encargado del proyecto. También está sometido, como no, a los cambios de requerimientos del cliente que a su vez están plenamente justificados por la readaptación de sus procesos de negocio a un mercado inestable.

Ante este cuadro de incertidumbre, el mayor desafio de una metodología de desarrollo es su *adaptación* para el cambio. Esto significa crear modelos que faciliten la *comunicación* entre todos los *agentes involucrados* en el sistema en construcción; que hagan visible la *trazabilidad* entre la concepción de los componentes, su especificación, implementación e instalación; significa el diseño de arquitecturas que faciliten la gestión de las dependencias entre estos componentes, que sean en fin, facilmente *reemplazables* por otros más optimizados o bien por componentes que aporten una mayor funcionalidad y/o facilidad de uso.

La dinámica de cambio no se desarrolla en la ingeniería del software con la misma velocidad vertiginosa con que nos tiene acostumbrados la tecnología del hardware. La clave reside en que a diferencia de la electrónica, en los dominios del desarrollo de software no existe un *vocabulario unificado*. Desde la fase de concepción de un sistema a la instalación de sus componentes hay que *mapear* entre sí una gran diversidad de lenguajes orientados al análisis, diseño, código ejecutable, esquemas de bases de datos, componentes de páginas web, entre otros. Esta distancia entre la concepción y la usabilidad de un producto o de un proceso de negocio, exige cada vez más la capacidad de cooperación y comunicación de un equipo interdisciplinar muy especializado. Esta guía visual de UML está pensada para facilitar este proceso cooperativo y para ayudar a establecer una buena práctica fundamentada en un lenguaje común.

¿A quién va dirigida esta guía visual?

Esta guía ha sido escrita y diseñada para los profesionales involucrados en todos los ciclos de actividad del desarrollo de sistemas de información (concepción, análisis y diseño, implementación, instalación de aplicaciones, gestión y certificación de proyectos); también para los que tengan responsabilidades en la especificación de procesos de negocio con el propósito de evaluar posibles reingenierías de procesos y/o diseño de bases de conocimiento; y finalmente, para aquellos equipos que estén inmersos en la preparación e implementación de certificaciones de calidad.

La claridad conceptual y los recursos didácticos utilizados en la exposición de los distintos procedimientos serán de utilidad para los estudiantes que sigan programas de autoaprendizaje y usen esta guía como complemento para sus lecturas de libros sobre UML. También los centros académicos y profesores dispondrán con esta guía de material interesante para completar sus diseños curriculares y proporcionar ejemplos prácticos a sus alumnos.

¿Cómo sacar un mayor provecho a su lectura?

La guía está organizada en unidades didácticas que describen la notación de los diagramas y las fuentes de información necesarias para definir los elementos de cada modelo. Puede usarse como consulta puntual de la notación de un diagrama, o bien, para revisar como establecer el hilo conductor entre los Casos de Uso (mapa funcional), las Clases de dominio (mapa conceptual), las Clases de Especifiación (types e interfaces) y las Clases de Implementación (código).

Un plan de estudio para realizar una progresiva asimilación de los conceptos podría empezar con los Casos de Uso (CU) y continuar con el análisis de los flujos de trabajo de un grupo de CU mediante los diagramas de Actividad; a continuación, separar los escenarios que agrupan una serie de actividades y hacer aflorar, a través de los diagramas de Interacción, los objetos que intercambian una serie de mensajes. A partir de este punto, disponemos del bagaje suficiente como para introducirnos en la abstracción de los objetos y comprender la importancia de separar las tres perspectivas básicas en nuestra representación de las clases: concepción, especificación e implementación. El siguiente paso es identificar alguna clase con un comportamiento complejo que la haga candidata a revisar todos sus posibles estados y averiguar que eventos son capaces de provocar un cambio de estado. El diagrama de Estados-Transición será el adecuado para representar esta dinámica de estados. Finalmente, abordaremos la configuración de componentes y su despliegue en una arquitectura.

Otra lectura de la guía puede estar mas centrada en el seguimiento de la metodología de desarrollo y la gestión de un proyecto. En este caso, las primeras secciones describen los niveles de concepción y formalización de un proyecto con la metodología TRAD (Taller de Requerimientos, Análisis y Diseño basado en el Proceso Unificado de Rational), y se van introduciendo progresivamente los diagramas y actividades que configuran la *unidad mínima de documentación* sostenible para un proyecto concreto.

El estudiante más avanzado podrá sacar también provecho con la consulta puntual de los diagramas en que esté más interesado y la revisión de sus extensiones. Las materias expuestas en las distintas secciones están actualizadas constantemente y pueden descargarse nuevas ediciones desde: http://www.vico.org/UMLguiavisual/

¿De dónde provienen las ideas expuestas?

El contenido de esta guía ha sido elaborado a partir del trabajo de una serie de profesionales que el autor ha tenido la oportunidad de estudiar y aplicar en distintos proyectos. Desde principios de los 90, los artículos publicados en el ***Journal of Object Oriented Programming*** (JOOP) por **James Odell**, **James Rumbaugh**, **Grady Booch**, **Desmond d'Souza**, **Bertrand Meyer**, **Steve Cook**, **John Daniels**, **Sally Shlaer** y **Stephen J. Mellor** entre otros, han sido una constante fuente de conocimiento. Publicaciones pioneras como el *Object Oriented Technology, A Manager's Guide* de **David A. Taylor**, en su primera edición de 1990 y en la segunda ampliada de 1998, han tenido una gran influencia en como abordar la presentación didáctica. También los libros de **Peter Coad** et al, *Object Oriented Analysis, Design and Programming, Object Models y Java Modeling Color with UML*, han sido de una ayuda extraordinaria. La obra enciclopédica *The Unified Modeling Language: Reference Manual* de **Rumbaugh & Jacobson & Booch**, es un punto de referencia constante. Sin duda, uno de los autores más influyentes ha sido **Martin Fowler**. Su primer libro *Analysis Patterns* continua siendo una referencia clave. Posteriormente, la primera edición de *UML Distilled* en 1997 y su última edición ampliada en 2000, se ha convertido en el libro de cabecera de UML. Otro clásico por la excelencia de su trabajo es *Applying UML and Patterns* de **Craig Larman** que en su segunda edición aparecida en verano de 2001 se ha superado a si mismo. También recientes y con muy buen material que ha sido incorporado a la guía, tenemos los libros de **Wendy & Michael Boggs**, *Mastering UML with Rational Rose*, de **Alistair Cockburn**, *Writing Effective Use Cases*; de **Scott W. Ambler**, *The Object Primer* segunda edición; y de **John Chessman & John Daniels**, *UML Components*, una de las novedades más interesantes de 2001. En la bibliografía sobre UML publicada en http://www.vico.org hay una relación completa de los libros consultados que se actualiza periódicamente con las últimas novedades.

Competencia y actuación

En los últimos veinte años de mi carrera profesional en el desarrollo de sistemas de información he participado en una gran diversidad de proyectos con distintos grados de responsabilidad e involucración, pero siempre con un compromiso firme en la calidad y usabilidad del producto final.

- **Entorno industrial**
  - CIM para la extrusión de polietileno y fabricación de mallas agrícolas y de embalaje.
  - CIM para el fraccionamiento de hemoderivados
  - Plan Funcional para la implementación SAP-Logística

- **Entorno sanitario**
  - Gestión de Bancos de Sangre y Hemoterapia
  - Planificación y gestión de campañas de captación de donantes
  - Gestión mutual de prestaciones asistenciales
  - Tarjeta Sanitaria para certificar transacciones asistenciales
  - Automatización de autoanalizadores de laboratorios de análisis
  - Integración de peticiones analíticas multicentricas y publicación de resultados
  - Gestión de laboratorios farmacéuticos
  - Historia Clínica Orientada por Problemas Automatizada
  - Libreta de Salud para programación de citas y exploraciones
  - Gestión integrada de servicios de Atención Primaria
  - Plan Funcional para la implementación SAP-Gestión Clínica y Asistencial

- **Entorno de ingeniería del software**
  - Framework de clases de análisis para definir mapas conceptuales
  - Framework de servicios comunes para la publicación dinámica de páginas HTML
  - Framework de certificación de entregables

- **Entorno administrativo y de gestión**
  - Plan Funcional para la implementación SAP-Contabilidad
  - Cuadro de control de indicadores de actividad y calidad
  - Sistema de información Ejecutivo

- **Entorno comercial**
  - Merchandising de productos farmacéuticos
  - Subastas y liquidación de lotes
  - Gestión de redes de puntos de venta con videoconferencia

- **Entorno de servicios**
  - Auditorías Informáticas
  - Plan de Sistemas de Información
  - Integración de sistemas de información de contabilidad administrativa y general

- **Entorno académico**
  - Programa de acceso a la universidad para mayores de 25 años
  - Gestión de títulos universitarios
  - Estudios de tercer cliclo: diseño curricular, publicación y gestión académica

- **Entorno docente**
  - Tutorías de proyectos de fin de carrera con UML
  - Cursos de Análisis, Diseño y Patrones
  - Talleres de introducción a UML
  - Talleres avanzados de UML con Rose y Visio
  - Talleres monográficos (Casos de Uso, Métrica, Metodología)

- **Entorno de I+D**
  - Bases de conocimiento con vocabularios controlados
  - Procesador de lenguaje natural dentro del dominio médico
  - Reconocimiento automático de conceptos clínicos a partir de textos no estructurados

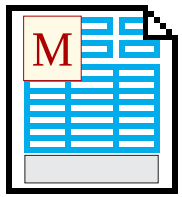
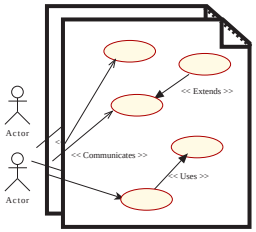
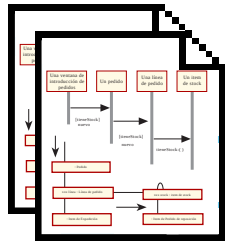
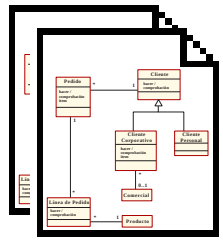
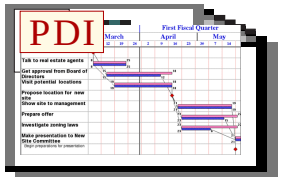
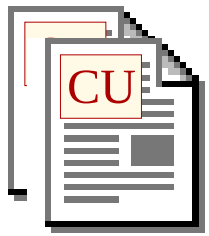
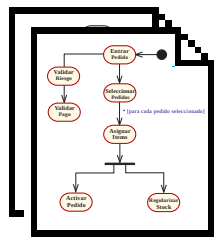
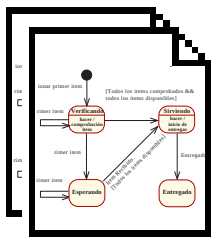
Agradecimientos

En primer lugar a los clientes que con su confianza han permitido que pueda desarrollar mi carrera profesional. También a los autores antes mencionados por su valioso trabajo en el avance de la disciplina de la ingeniería del software y en la consolidación de UML como *lingua franca*.

Josep Vilalta
Badalona, Barcelona (España)
jvilalta@vico.org
http://www.vico.org

vico.org

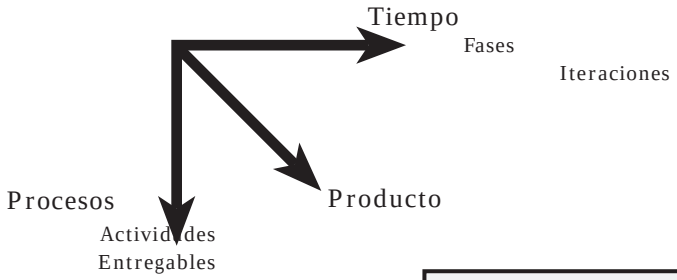
Niveles de concepción y formalización de un proyecto

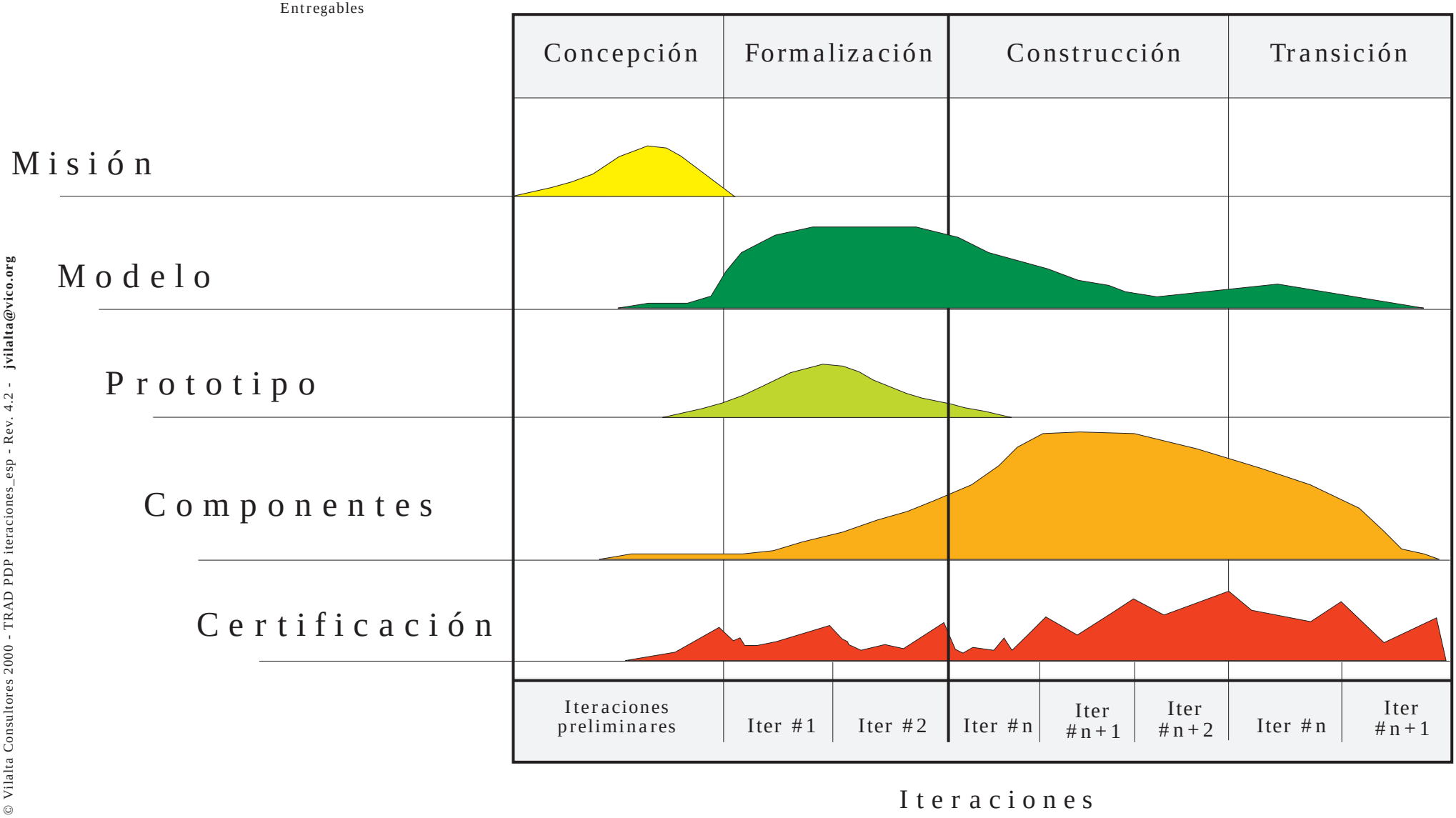
Nivel Ø	Nivel 1	Nivel 2	Nivel 3	Nivel 4	Nivel 5
					
					
Matrícula del Proyecto	Glosario de Conceptos	Funcionalidad Diagramas de Casos de Uso	Escenarios Interacción de objetos	Clases Análisis Diseño Implementación	
					
"Code like hell"					
					
Cronograma Plan Director Iteraciones	Procesos Principales	Especificación Casos de Uso	Flujos de Trabajo	Dinámica Eventos - Estados	

Código

PDP

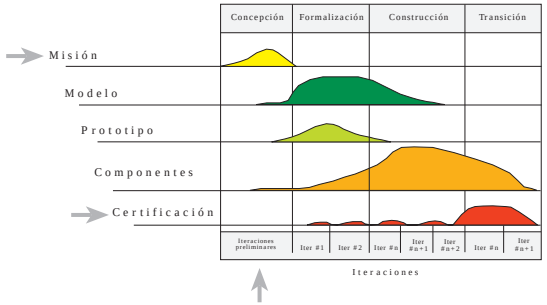
Esfuerzo de desarrollo



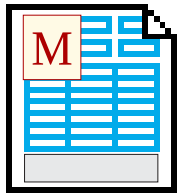


P D P

Plan Director de Proyecto

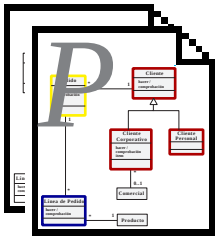


Concepción



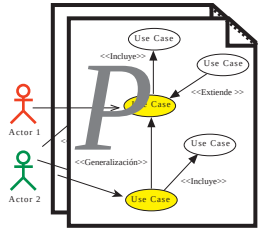
Matrícula del proyecto

Procesos principales



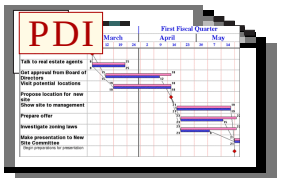
Patrones de Análisis

Misión



Patrones de Funcionalidad

Glosario de Conceptos

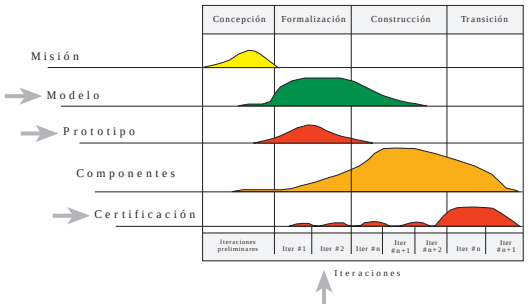


Cronograma Plan Director Iteraciones

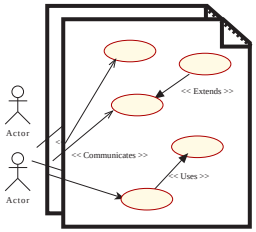
Anteproyecto

P D P

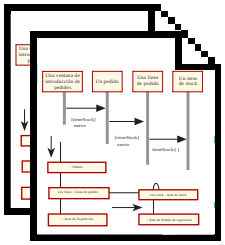
Plan Director de Proyecto



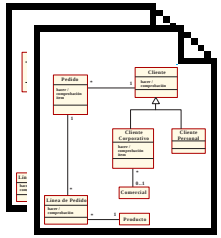
Formalización



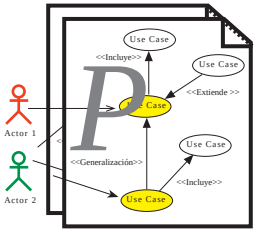
Funcionalidad
Diagramas de Casos de Uso



Escenarios
Interacción de objetos

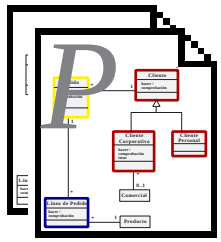


Clases
Análisis y Diseño

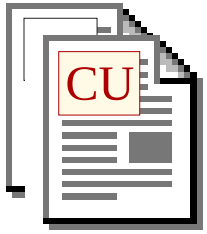


Patrones de Funcionalidad

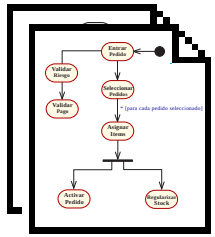
Modelo



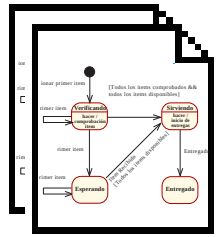
Patrones de Análisis



Especificación Casos de Uso



Flujos de Trabajo

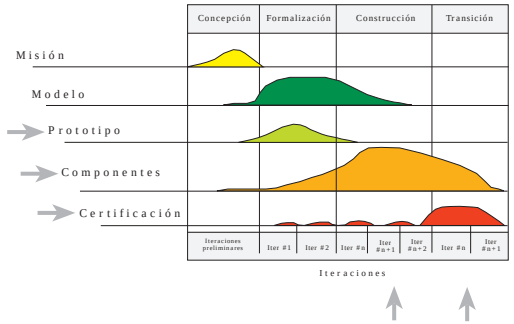


Dinámica Eventos Estados

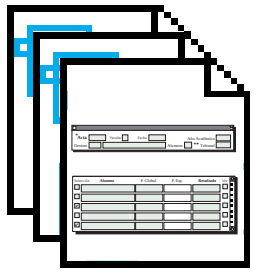
vico.org

P D P

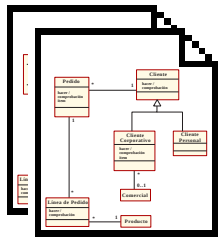
Plan Director de Proyecto



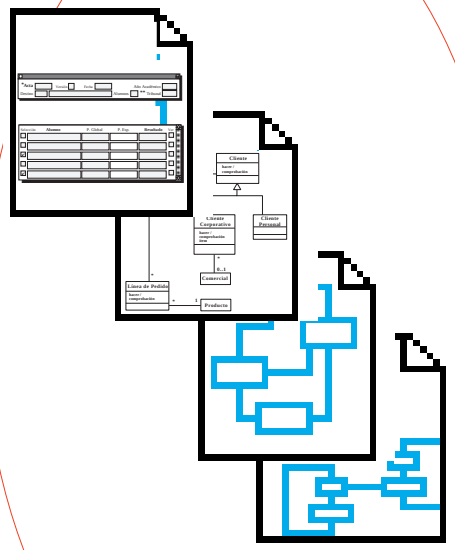
Construcción



Interface
Gráfico de Usuario

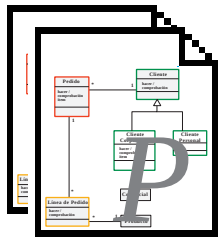


Clases
Diseño
Implementación

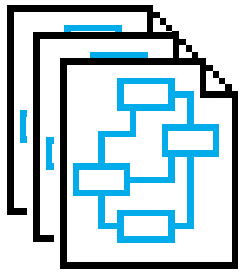


Framework
de Aplicaciones

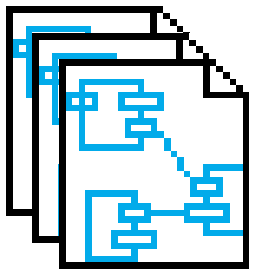
Componentes



Patrones de
Diseño



Base de Datos
Esquema de Persistencia

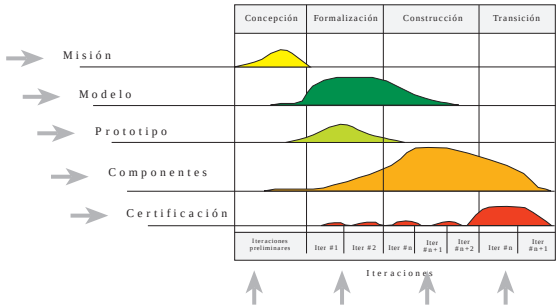


Arquitectura
Componentes

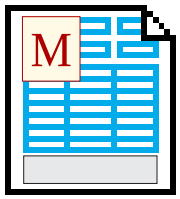
vico.org

P D P

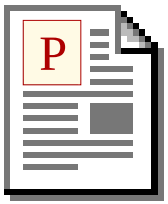
Plan Director de Proyecto



Concepción

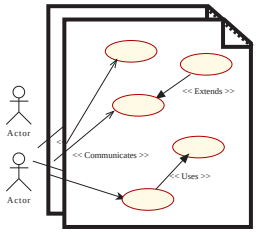


Matrícula del proyecto

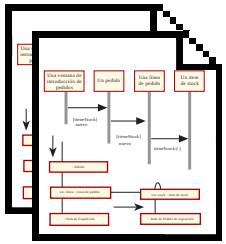


Procesos principales

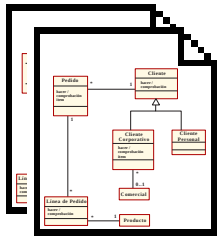
Formalización



Funcionalidad
Diagramas de Casos de Uso

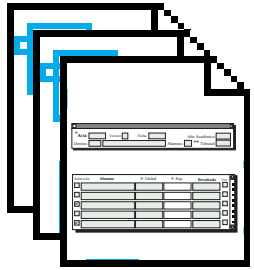


Escenarios
Interacción de objetos

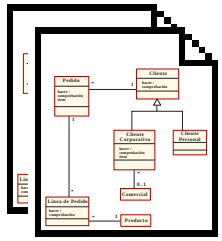


Clases
Análisis y Diseño

Construcción



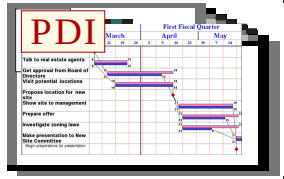
Interface Gráfico de Usuario



Clases
Diseño Implementación

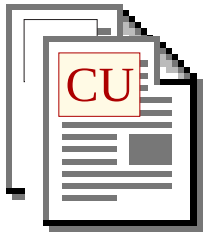
Misión

Glosario de Conceptos

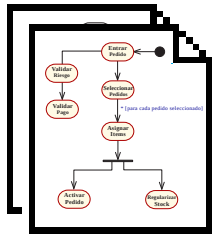


Cronograma Plan Director Iteraciones

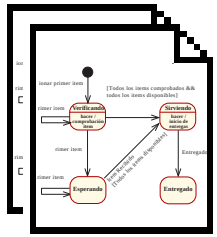
Modelo



Especificación Casos de Uso

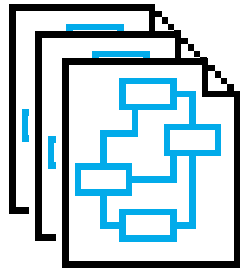


Flujos de Trabajo

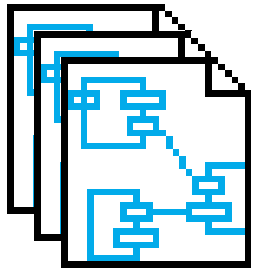


Dinámica Eventos Estados

Componentes



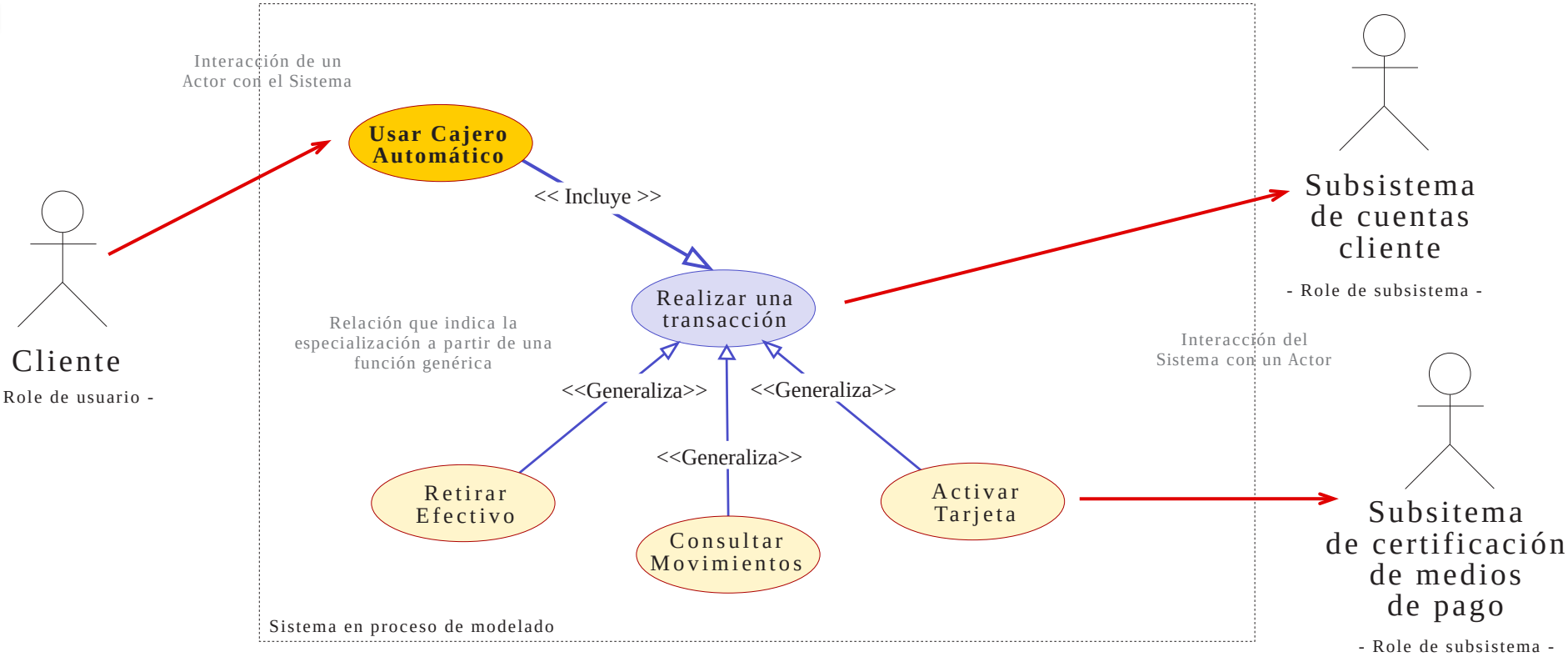
Base de Datos
Esquema de Persistencia



Arquitectura
Componentes

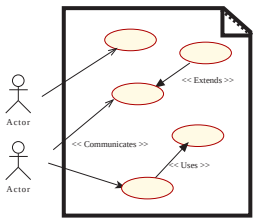
Anteproyecto

¿ Cómo identificar Actores ?



Actores

- Representan a un agente que interactúa con el sistema
- No son parte del sistema que se desarrolla
- Entran información al sistema
- Reciben información del sistema
- Entran y reciben información

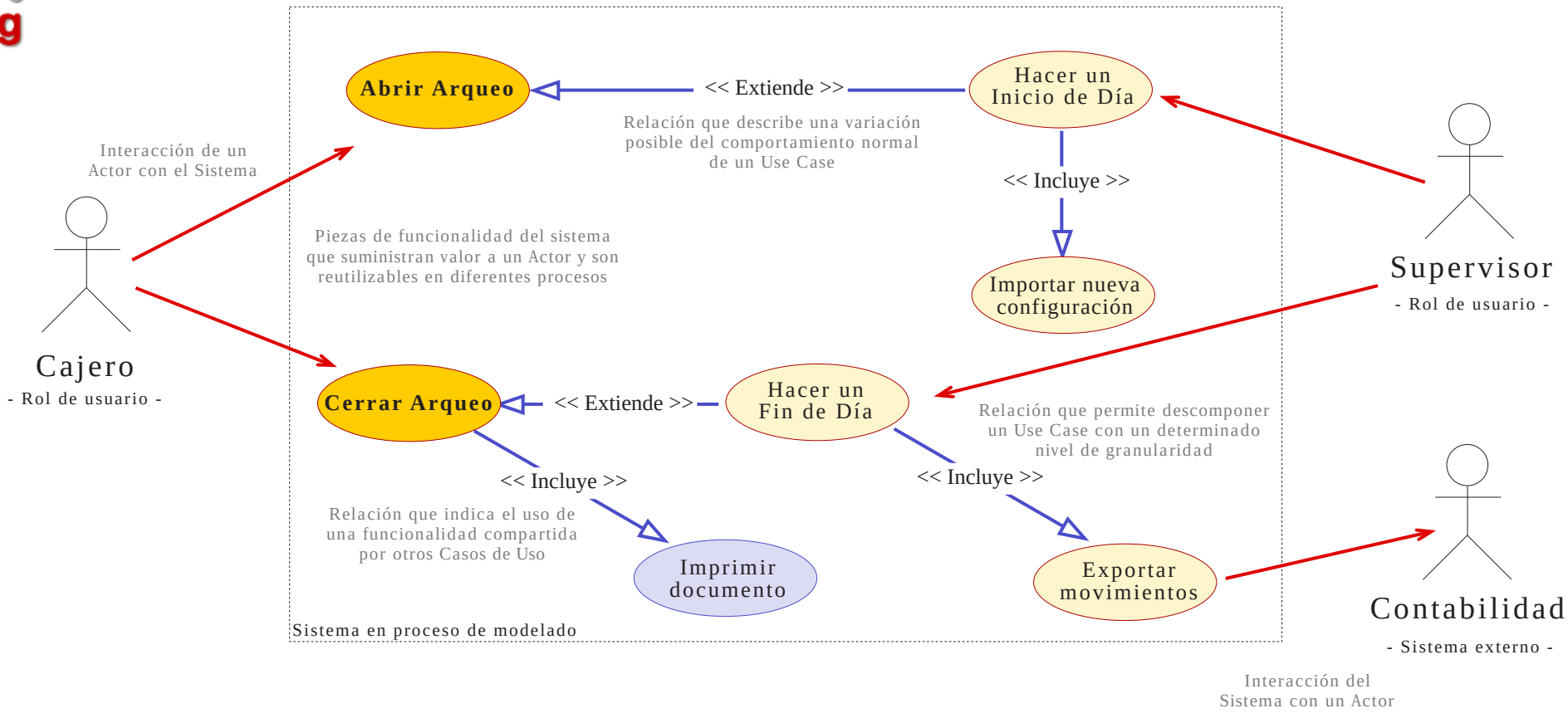


Funcionalidad
Diagramas de Casos de Uso

A la búsqueda de Actores.-

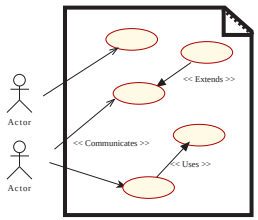
1. ¿ Quien está interesado en un requerimiento concreto ?
2. ¿ En qué dominios de la organización se usará el sistema ?
3. ¿ Quien será beneficiario de la nueva funcionalidad ?
4. ¿ Quien proveerá, usará y/o retirará, información ?
5. ¿ Quien dará soporte y administrará el sistema ?
6. ¿ Usará el sistema un recurso externo ?
7. ¿ Un usuario actuará con diferentes roles ?
8. ¿ Diferentes usuarios actuarán con un mismo rol ?
9. ¿ Interaccionará el nuevo sistema con un sistema antiguo ?

¿ Cómo identificar Casos de Uso ?



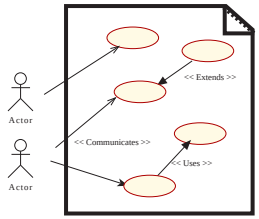
A la búsqueda de Casos de Uso.-

1. ¿ Cuales son las tareas y responsabilidades de cada actor ?
2. ¿ Algún actor creará, almacenará, cambiará, borrará o leerá información del sistema ?
3. ¿ Qué Casos de Uso crearán, almacenarán, cambiarán, borrarán o leerán esta información ?
4. ¿ Es necesario que un Actor informe al sistema sobre cambios externos ?
5. ¿ Es necesario que un Actor sea informado sobre ciertas incidencias del sistema ?
6. ¿ Qué Casos de Uso darán soporte y mantendrán el sistema ?
7. ¿ Pueden ser realizados por los Casos de Uso todos los requerimientos funcionales documentados ?



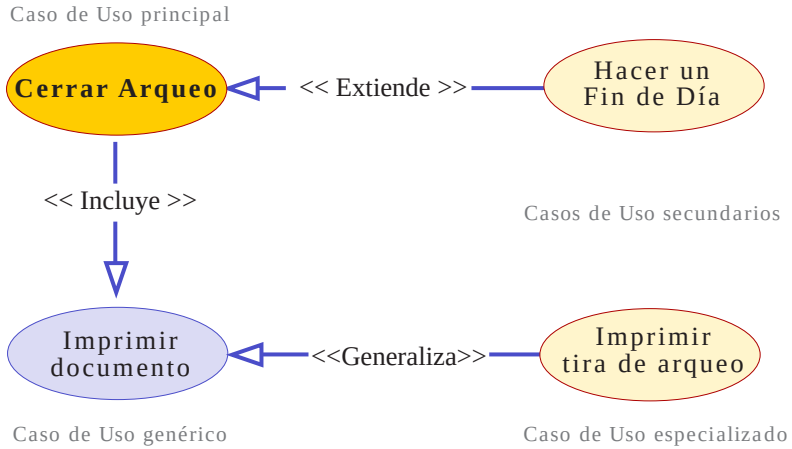
Funcionalidad
Diagramas de Casos de Uso

vico.org



Funcionalidad
Diagramas de
Casos de Uso

Especificación de un Caso de Uso



L Límites: Cuando empieza y cómo termina el Caso de Uso.

I Interacciones: Comportamiento de Actores y Sistema. Acción-Reacción dentro del Caso de Uso.

M Masa: Conjunto de Objetos e Interfaces que requiere el Caso de Uso.

I Índice de escenarios: Flujo principal de eventos y secuencia de variaciones posibles dentro de un Caso de Uso.

T Tribulaciones: Contingencias probables que pueden afectar al flujo de los eventos y son excepciones del Caso de Uso.

Propósito
- Regla de Negocio -

Cerrar un periodo de movimientos de caja para obtener una situación real de dinero en efectivo y en documentos.

Precondiciones

- **Arqueo abierto**
- **Actor habilitado**
- **TPV abierto**
- **TPV habilitado**

Activación

- **A discreción de un Actor habilitado**

Flujo Principal

1. Sistema *requiere confirmación de cierre* de arqueo.

2. Sistema *comprueba la configuración* de TPV para identificar tipo de cierre.

3. Sistema *calcula los totales teóricos* para cada forma de cobro.

4. Actor *introduce totales reales* para cada forma de cobro.

5. Sistema *registra el cierre*: importes reales para cada forma de cobro y descuadres.

6. Sistema *imprime el documento* "Tira de Arqueo".

Variaciones

a. Si Actor decide *aparcar* el cierre de arqueo, sistema activa UC Aparca_arqueo y termina el UC.

a. Cierre de *arqueo de tipo abierto*:
- Actor decide el momento de imprimir el documento "Tira de Arqueo".

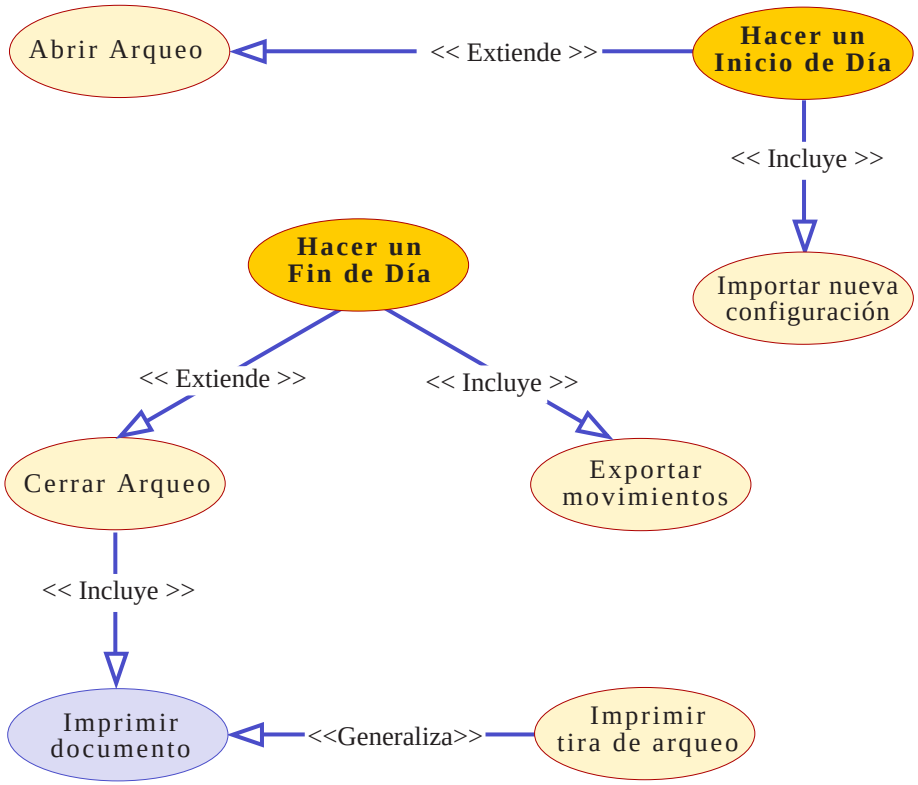
b. Cierre de *arqueo de tipo ciego*:
- Sistema no muestra los totales teóricos para cada forma de cobro.

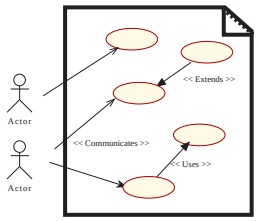
Excepciones

a. Si el servidor está off-line, *sistema informa al usuario*, termina el UC manteniendo el arqueo abierto.

a. Si impresora está off-line, *sistema informa al usuario* y le *pide confirmación* para continuar.

Ventajas del modelo Use Case





Funcionalidad
Diagramas de Casos de Uso

Piezas de funcionalidad reutilizables en diferentes procesos de negocio

1. Lenguaje de comunicación entre usuarios y desarrolladores.

2. Comprensión detallada de la funcionalidad del sistema.

3. Acotación precisa de las habilitaciones de los usuarios.

4. Gestión de riesgo más eficiente para gobernar la complejidad.

5. Estimación más exacta para determinar tiempo, recursos y prioridades en la dosificación de esfuerzo de desarrollo.

6. Fiel trazabilidad para verificar la traducción de requerimientos en código ejecutable.

7. Mayor control para mantener las sucesivas revisiones de los programas.

8. Certificación contractual Cliente-Desarrollador.

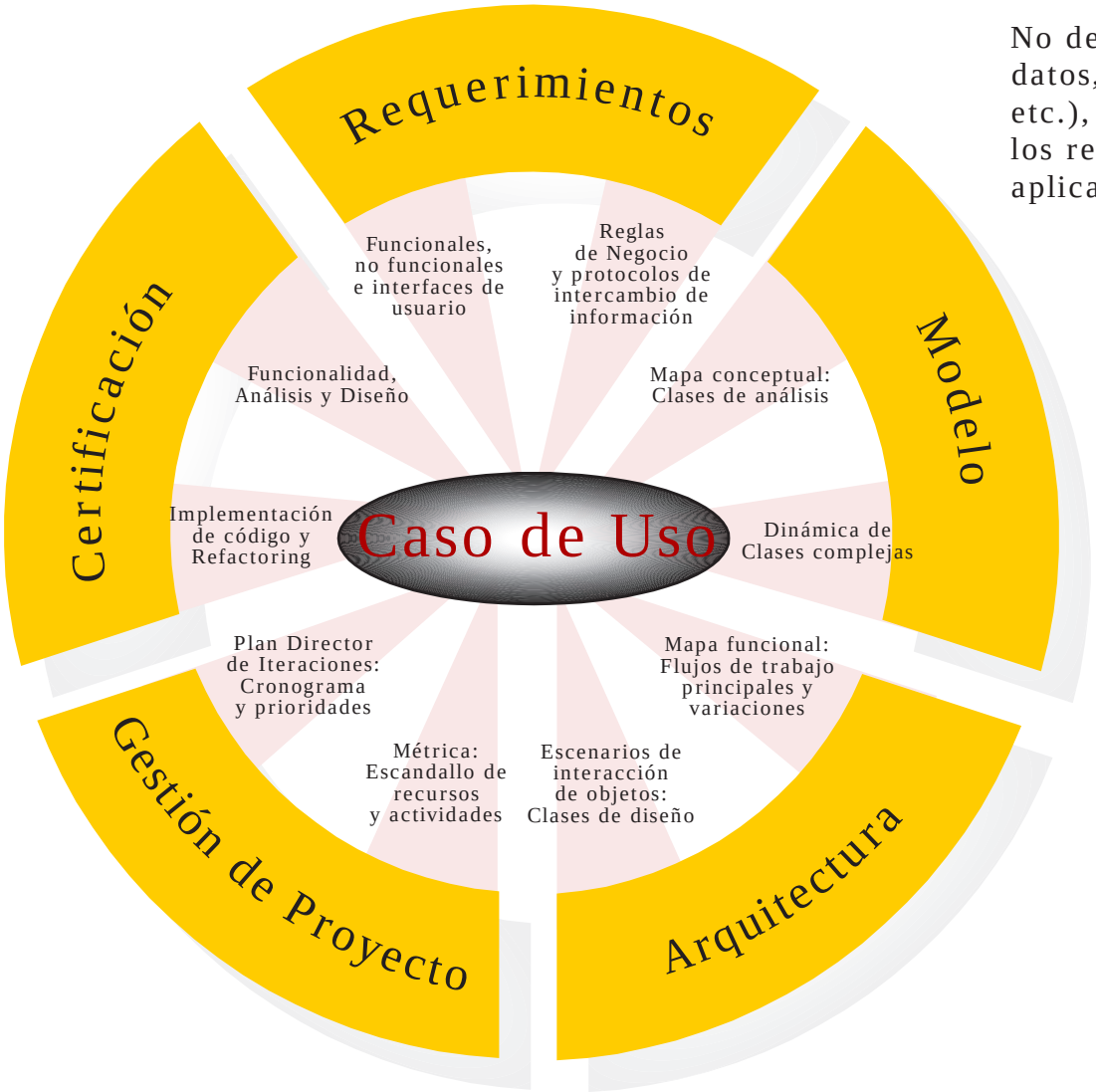
9. Documentación orientada al usuario: Helps - Manual de Procedimientos - Reglas de Negocio.

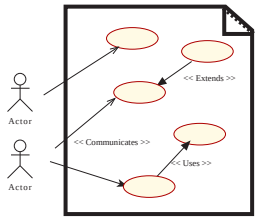
10. Documentación orientada al administrador del sistema: Soporte de Mantenimiento.

Requerimientos y Casos de Uso

Los Casos de Uso son requerimientos funcionales que describen de una manera detallada el comportamiento del sistema con los distintos Actores que interactúan con él.

No definen todos los requerimientos (por ej. los tipos de datos, interfaces externas, niveles de rendimiento esperado, etc.), pero representan el hilo conductor que vincula a todos los requerimientos posibles (actuales y futuros) de una aplicación.





Funcionalidad
Diagramas de Casos de Uso

CU Realizar pedido

Flujo Principal	Variaciones
1. Usuario *identifica el cliente* que ha enviado un pedido.	
2. Usuario *entra lineas de pedido*, con su código de artículo, tipo de presentación y cantidad.	
3. Sistema *comprueba cada línea* del pedido para validar la situación del artículo en catálogo y el número de items del artículo en stock.	**a.** Artículo NO está vigente en catálogo, sistema informa que *articulo no está vigente* y muestra artículos alternativos activando el **CU Seleccionar artículo**. **b.** SI existen suficientes items del artículo en el stock, sistema *asigna items al pedido*. **c.** NO existen suficientes items del artículo en stock, o la asignación de items deja la situación del artículo en stock por debajo del nivel de reposición, sistema *genera pedido de reposición* activando el **CU Generar pedido**.
4. Sistema *comprueba la situación del pedido* .	**a.** SI se ha realizado el pago y SI todos los items del pedido han sido asignados, sistema informa que procede a *servir el pedido* activando el **CU Servir pedido**. **b.** SI se ha realizado el pago y NO existen suficientes items del artículo en stock, sistema *aparca el pedido* del cliente activando el **CU Aparcar pedido**. **c.** SI no se ha realizado el pago según el plazo convenido, sistema *cancela el pedido*.

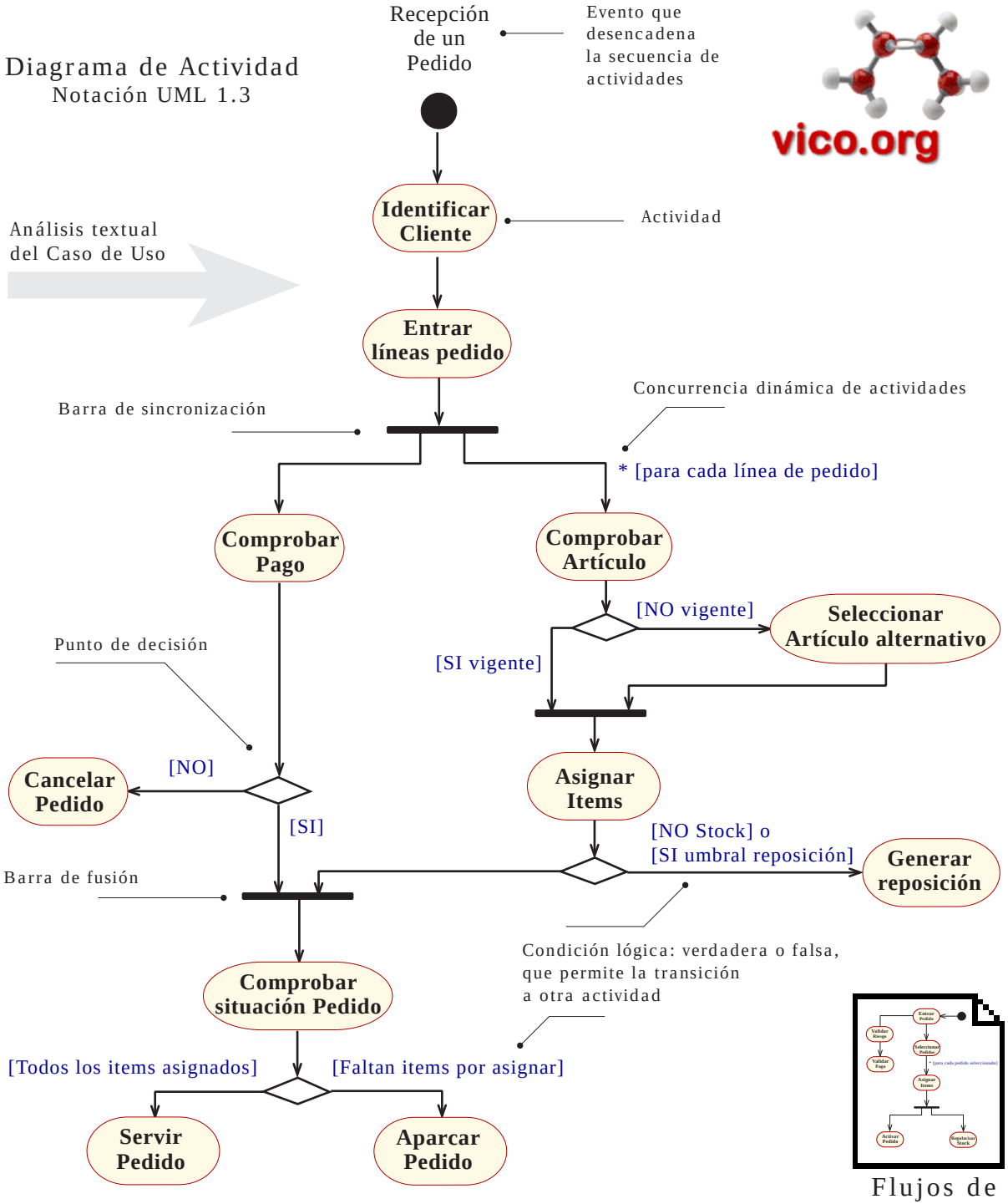
1

Descripción de un Flujo de Trabajo

Un flujo de trabajo muestra la secuencia de actividades que se desarrollan dentro de uno o varios Casos de Uso, como una pieza de funcionalidad concreta que satisface los requerimientos de un Actor.

Diagrama de Actividad
Notación UML 1.3

Análisis textual del Caso de Uso



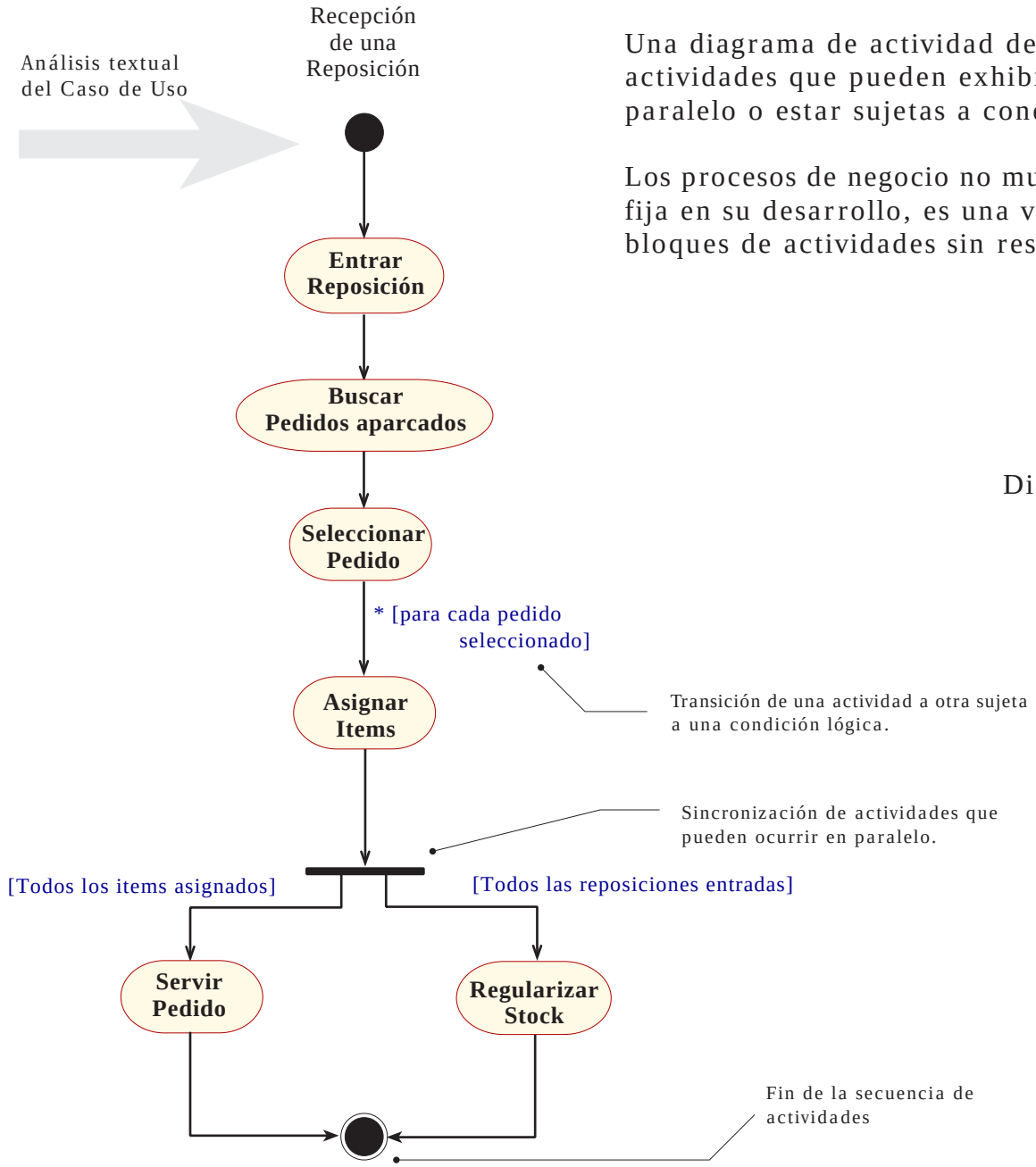
Flujos de Trabajo

CU Actualizar reposición

Flujo Principal

1. Usuario *recepciona albaranes* de reposición que ha enviado un proveedor.
2. Sistema *localiza* los pedidos de clientes aparcados que pueden cumplimentarse con la nueva entrada de items.
3. Usuario *selecciona* los pedidos de clientes aparcados que decide cumplimentar.
4. Sistema *asigna* los items pendientes a los pedidos de cliente seleccionados.
5. Sistema informa que procede a *servir el pedido* activando el **CU Servir pedido**..
6. Sistema *regulariza* la situación de items en stock y revisa los umbrales de reposición automática.

Análisis textual del Caso de Uso

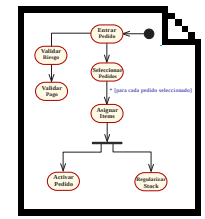


Descripción de un Flujo de Trabajo

Una diagrama de actividad describe una secuencia de actividades que pueden exhibir un comportamiento en paralelo o estar sujetas a condiciones lógicas.

Los procesos de negocio no muestran siempre una secuencia fija en su desarrollo, es una ventaja así poder modelar bloques de actividades sin restricciones de concurrencia.

Diagrama de Actividad
Notación UML 1.3



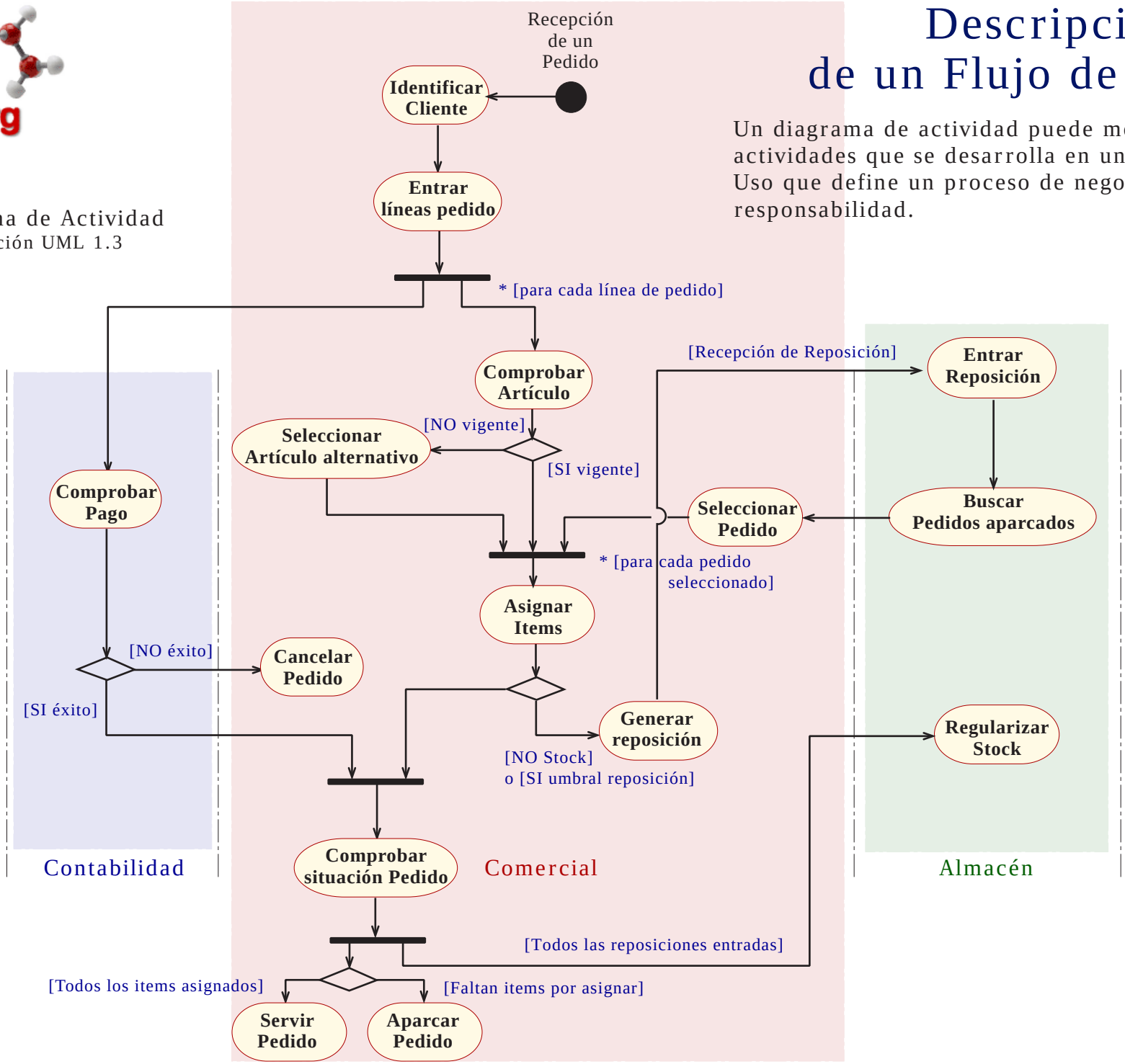
Flujos de Trabajo

vico.org

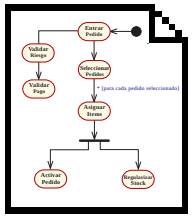
Descripción de un Flujo de Trabajo

Un diagrama de actividad puede mostrar la secuencia de actividades que se desarrolla en un paquete de Casos de Uso que define un proceso de negocio y sus áreas de responsabilidad.

Diagrama de Actividad
Notación UML 1.3



Líneas para acotar áreas de responsabilidad (swim-lines)



Flujos de Trabajo

Descripción de un Flujo de Trabajo

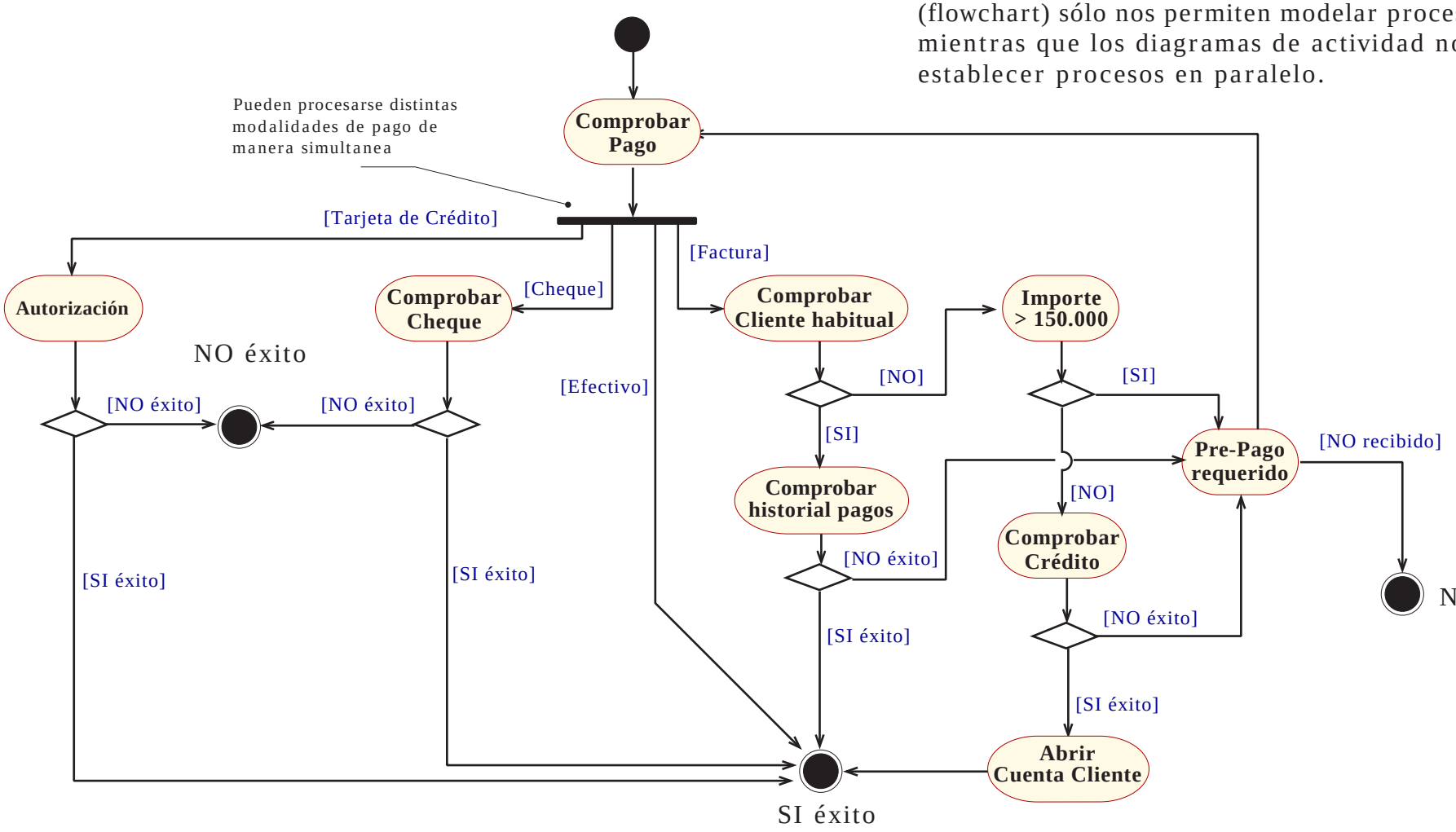
Descomposición de la actividad

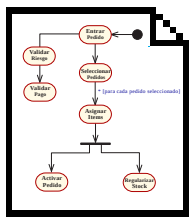
Comprobar Pago

Diagrama de Actividad
Notación UML 1.3

Su objetivo no es relacionar actividad con objetos, sólo comprender qué actividades son necesarias y cuales son sus relaciones de dependencia.

El diagrama de actividad nos permite definir en qué orden se van a realizar distintas tareas. Los diagramas de flujo (flowchart) sólo nos permiten modelar procesos secuenciales, mientras que los diagramas de actividad nos permiten establecer procesos en paralelo.





Flujos de Trabajo

CU Realizar pedido

Flujo Principal	Variaciones
1. Usuario *identifica el cliente* que ha enviado un pedido.	
2. Usuario *entra lineas de pedido*, con su código de artículo, tipo de presentación y cantidad.	
3. Sistema *comprueba cada línea* del pedido para validar la situación del artículo en catálogo y el número de items del artículo en stock.	**a.** Artículo NO está vigente en catálogo, sistema informa que *articulo no está vigente* y muestra artículos alternativos activando el **CU Seleccionar artículo**. **b.** SI existen suficientes items del artículo en el stock, sistema *asigna items al pedido*. **c.** NO existen suficientes items del artículo en stock, o la asignación de items deja la situación del artículo en stock por debajo del nivel de reposición, sistema *genera pedido de reposición* activando el **CU Generar pedido**.

Análisis textual del Use Case

Objetos que interactúan

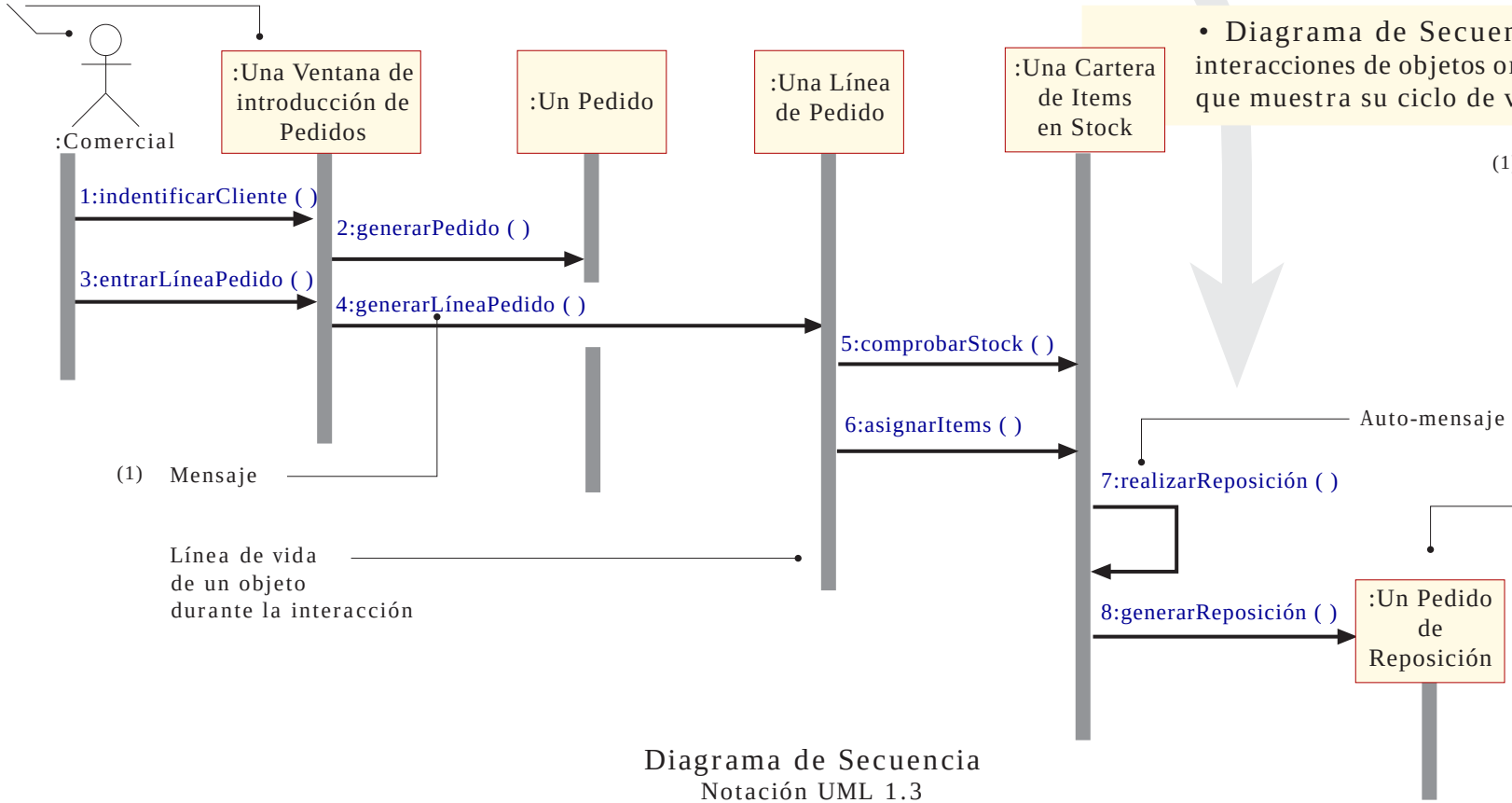


Diagrama de Secuencia
Notación UML 1.3

Descripción de un Escenario

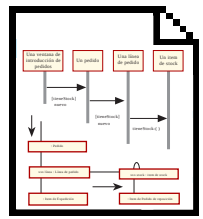
Un escenario muestra de que manera interactúan los distintos objetos dentro del flujo principal de eventos de un Caso de Uso con alguna variación o extensión concreta del mismo.

Utilizamos dos diagramas de interacción:
a/. Secuencia
b/. Colaboración
Su finalidad es describir los mensajes que intercambian los distintos objetos para cumplir con las responsabilidades definidas en un escenario concreto de un Caso de Uso.

• Diagrama de Secuencia.- Representa las interacciones de objetos ordenadas en una serie temporal que muestra su ciclo de vida.

(1) Mensajes enviados entre los objetos descritos en el flujo de eventos de un Caso de Uso.

Estos mensajes muestran el nivel de colaboración entre los distintos objetos existentes e indican cuando se requiere generar nuevos objetos para cumplir con las responsabilidades asignadas.



Escenarios

Descripción de un Escenario

Con un escenario representamos el conjunto de eventos que configura el comportamiento de un Caso de Uso.

Un escenario describe una instancia del flujo de eventos de un Caso de Uso, con sus variaciones o extensiones posibles y las excepciones probables.

Utilizamos dos diagramas de interacción:
a/. Secuencia
b/. Colaboración
Su finalidad es describir los mensajes que intercambian los distintos objetos para cumplir con las responsabilidades definidas en un escenario concreto de un Caso de Uso.

• Diagrama de colaboración.- Representa una posible interacción de los objetos ordenados a partir de la topología que muestra el envio de sus mensajes.

CU Realizar pedido

Flujo Principal	Variaciones
1. Usuario *identifica el cliente* que ha enviado un pedido.	
2. Usuario *entra lineas de pedido*, con su código de artículo, tipo de presentación y cantidad.	
3. Sistema *comprueba cada línea* del pedido para validar la situación del artículo en catálogo y el número de items del artículo en stock.	**a.** Artículo NO está vigente en catálogo, sistema informa que *articulo no está vigente* y muestra artículos alternativos activando el **CU Seleccionar artículo**. **b.** SI existen suficientes items del artículo en el stock, sistema *asigna items al pedido*. **c.** NO existen suficientes items del artículo en stock, o la asignación de items deja la situación del artículo en stock por debajo del nivel de reposición, sistema *genera pedido de reposición* activando el **CU Generar pedido**.

Análisis textual del Use Case

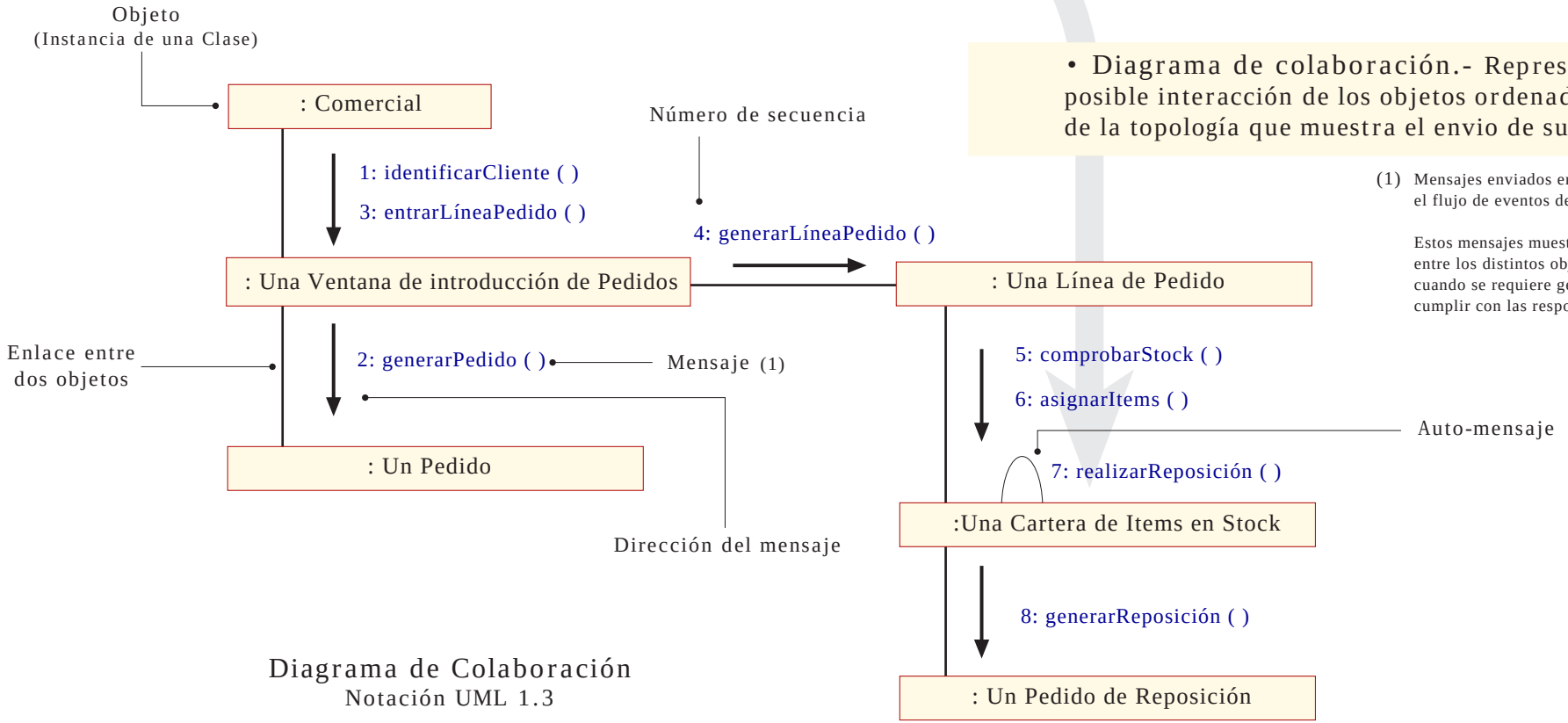
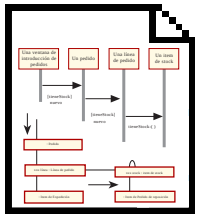


Diagrama de Colaboración
Notación UML 1.3

(1) Mensajes enviados entre los objetos descritos en el flujo de eventos de un Caso de Uso.

Estos mensajes muestran el nivel de colaboración entre los distintos objetos existentes e indican cuando se requiere generar nuevos objetos para cumplir con las responsabilidades asignadas.



Escenarios

Clases

Desde una perspectiva conceptual, una Clase representa un conjunto de Objetos que comparten:

- Las mismas propiedades (Atributos)
- El mismo comportamiento (Métodos)
- Las mismas relaciones con otros Objetos (Mensajes)
- La misma semántica dentro del sistema

Desde una perspectiva física, una Clase es una pieza de software que actua como un molde para fabricar tipos particulares de objetos que disponen de los mismos atributos y métodos.

Estos elementos que configuran cada tipo de objeto sólo se definen una vez, cuando especificamos la estructura de la Clase a la que pertenecen.

Los objetos que se han creado a partir de una Clase concreta, se llaman instancias de esta Clase y se diferencian entre ellos únicamente por los valores de sus atributos (variables).

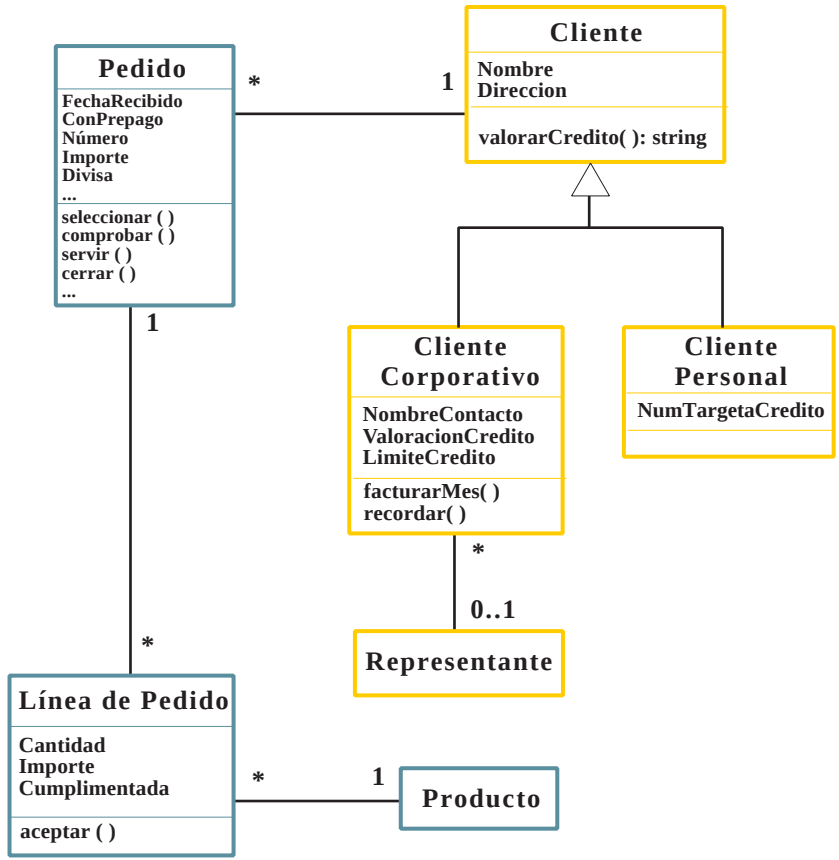


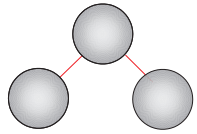
Diagrama de Clases
Notación UML 1.3

Objetos

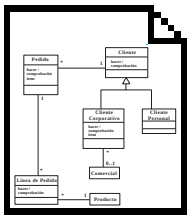
Un Objeto representa una entidad del mundo real o inventada. Es un concepto, una abstracción o algo que dispone de unos límites bien definidos y tiene una significación para el sistema que se pretende modelar.

Estructura y función:

- Identidad ¿Quien soy? = Atributos
- Propósito ¿Cual es mi misión? = Justificación
- Responsabilidades ¿Qué debo hacer? = Métodos
- Procedencia ¿De qué Clase provengo? = Origen
- Relaciones ¿Qué mensajes entiendo? = Comportamiento



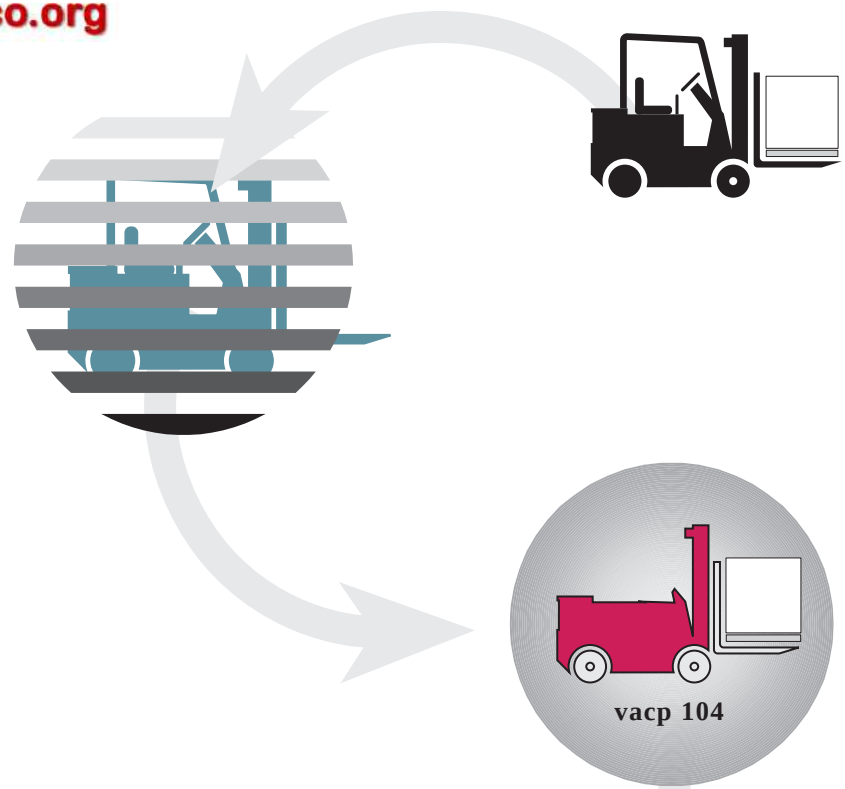
Objetos

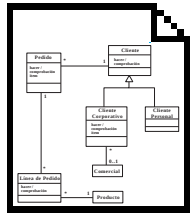


Clases

Abstracción

A partir de una abstracción del mundo real, definimos objetos que representan micromódulos de software ideales. Desde su creación, se mantienen de manera independiente unos de otros, sólo interaccionan con otros objetos a través de sus mensajes. Cada objeto configura un universo ordenado y autosuficiente.



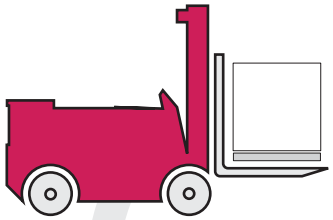


Clases

Objeto

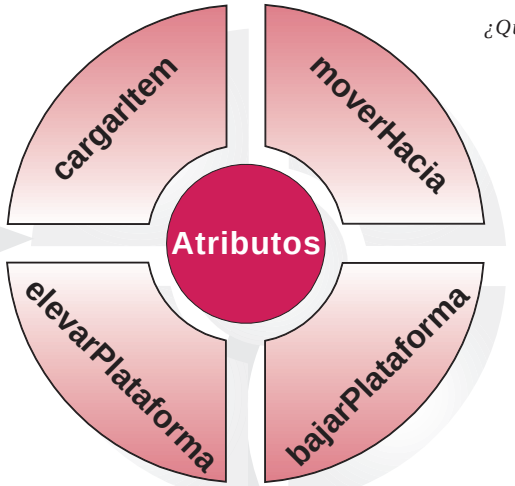
Todo lo que un objeto “conoce” está representado en sus *atributos* (variables y estado actual), y todo lo que puede “realizar” está definido en sus *métodos* (comportamiento), y en sus “interacciones” con otros objetos a través del intercambio de *mensajes* (dinámica del ciclo de vida).

¿Quien soy?



Vehículo Automático Carga Palets
vacp104

¿Qué puedo hacer?



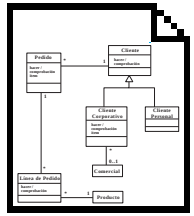
¿Qué conozco?

Variables.-

- **Identificación**
- **Medidas de la carga**
- **Capacidad de carga**
- **Velocidad máxima**
- **Tipo de carga**

Estado.-

- **Localización**
- **Orientación**
- **Velocidad**

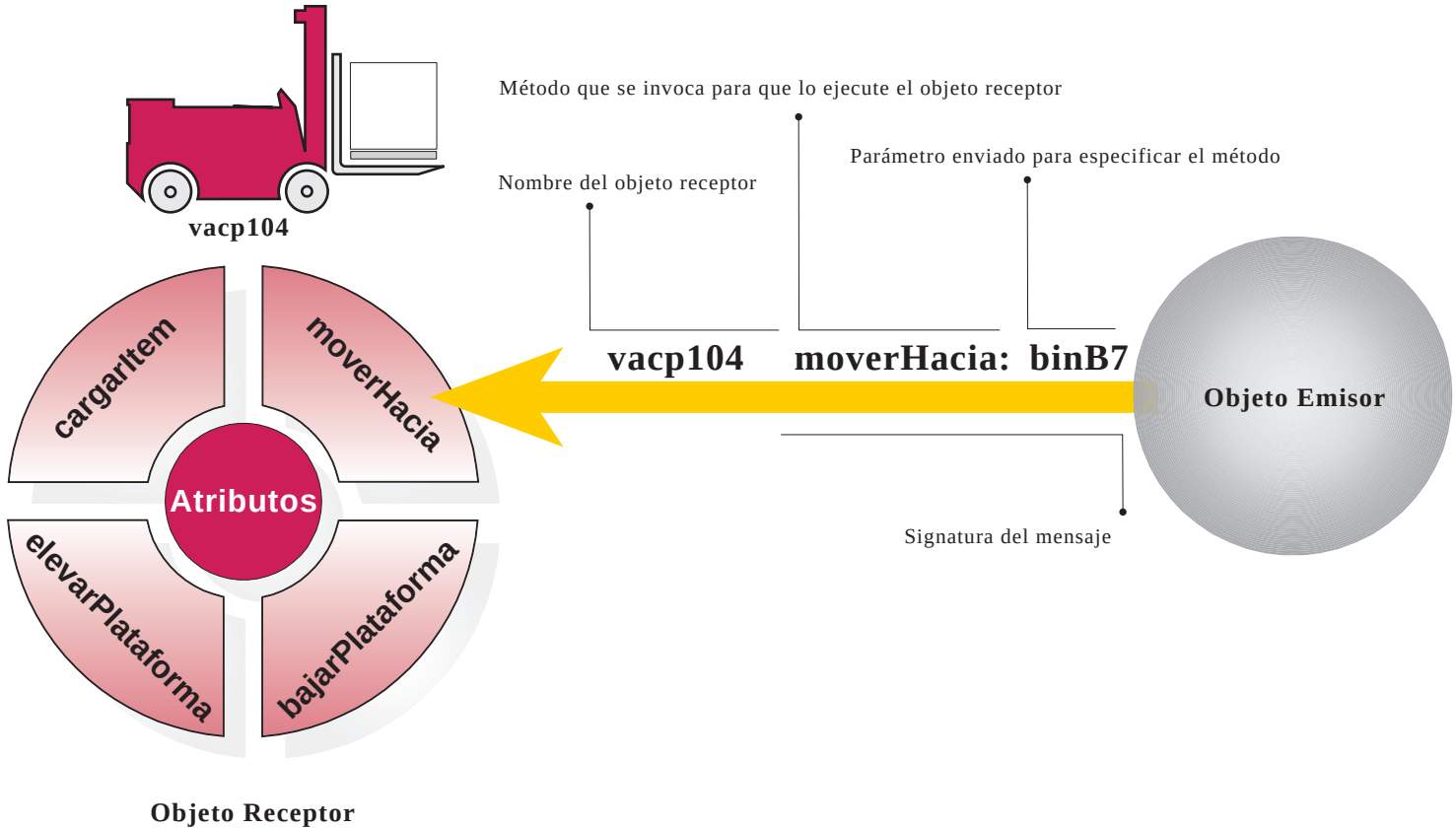


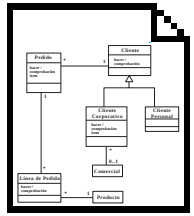
Clases

Mensaje

Una aplicación orientada a objetos consiste en un número determinado de objetos que interactuan entre si enviándose mensajes unos a otros para invocar sus métodos. Este intercambio de mensajes facilita su comportamiento, cambios de estado, destrucción o almacenamiento.

Ya que todo lo que un objeto puede realizar está expresado en sus métodos, este simple mecanismo de mensajes soporta todas las posibles interacciones entre ellos.





Clases

Encapsulación

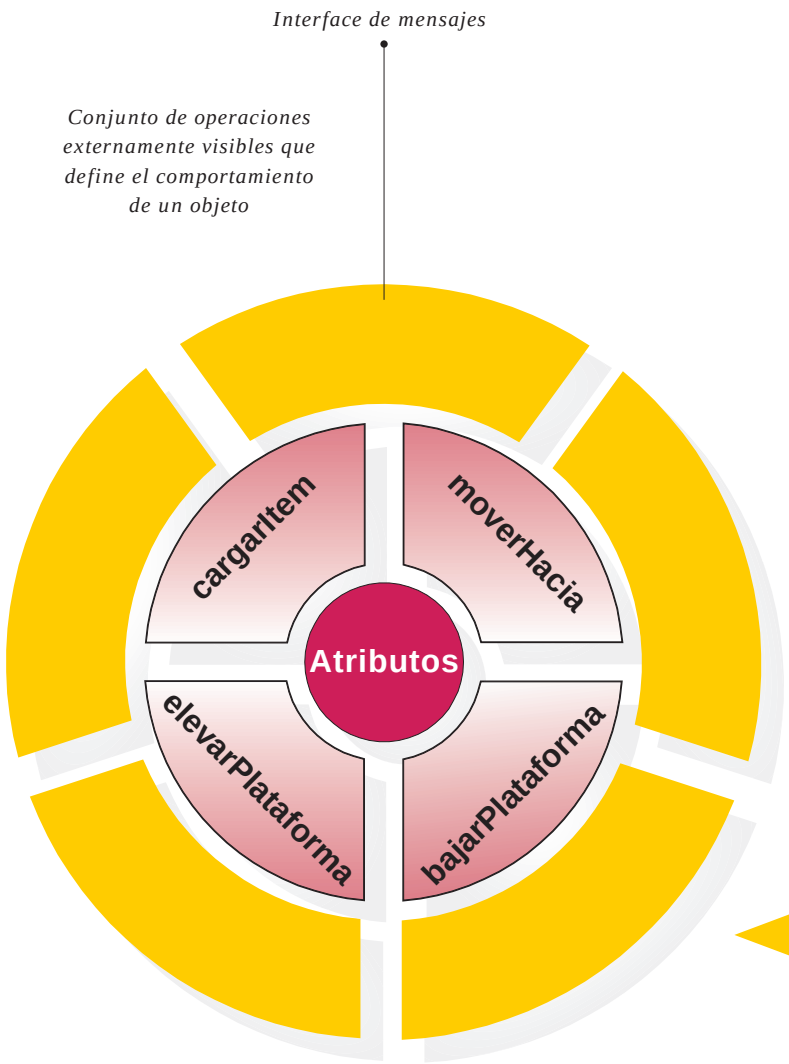
El empaquetado de métodos y atributos dentro de un objeto mediante una interface de mensajes, es lo que denominamos encapsulación.

La clave está precisamente en este envoltorio del objeto. La interface que rodea por completo al objeto actúa como punto de contacto para todos los mensajes que llegan desde cualquier objeto emisor.

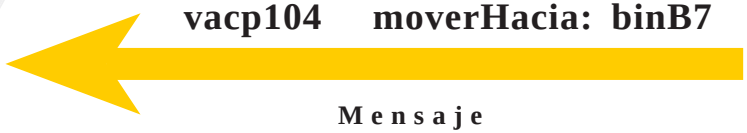
La interface de mensajes tiene la misma función que la membrana de una célula, disponer de una barrera esencial entre la estructura interna de un objeto y el exterior.

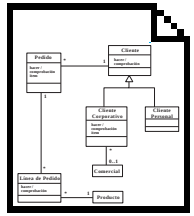
Su propósito es garantizar que todas las interacciones del objeto tengan lugar a través de un sistema de mensajería predefinido con el que es capaz de entenderse y reaccionar adecuadamente.

No existe ninguna otra manera con la que un objeto externo pueda acceder a los atributos y métodos escondidos dentro de la interface de mensajes de otro objeto.



vacp104



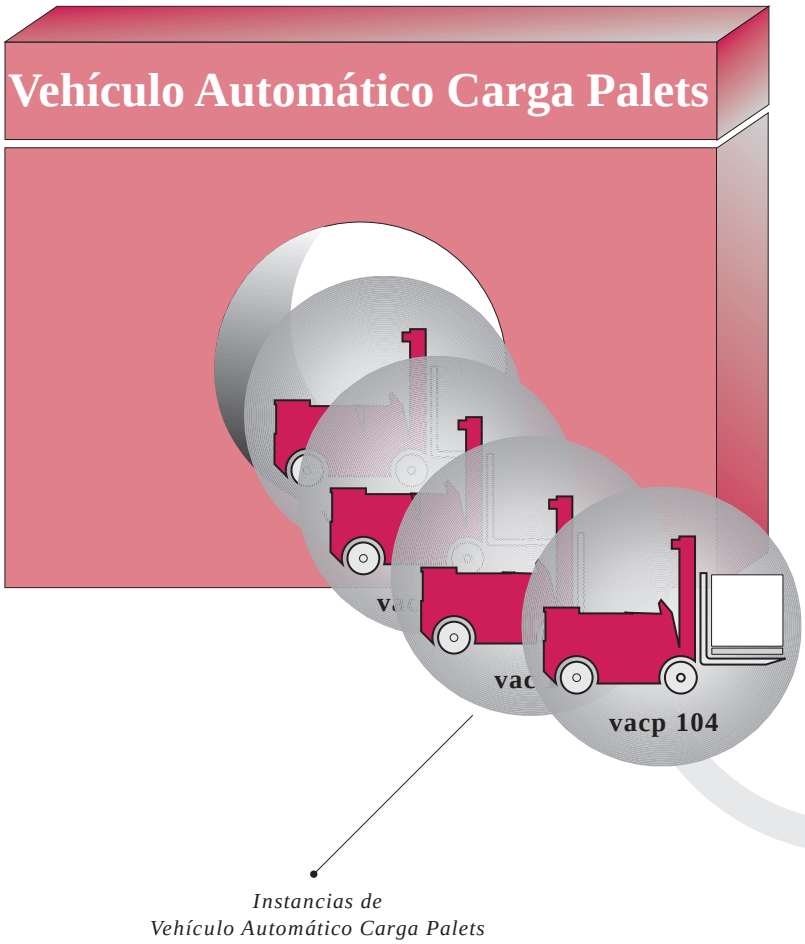


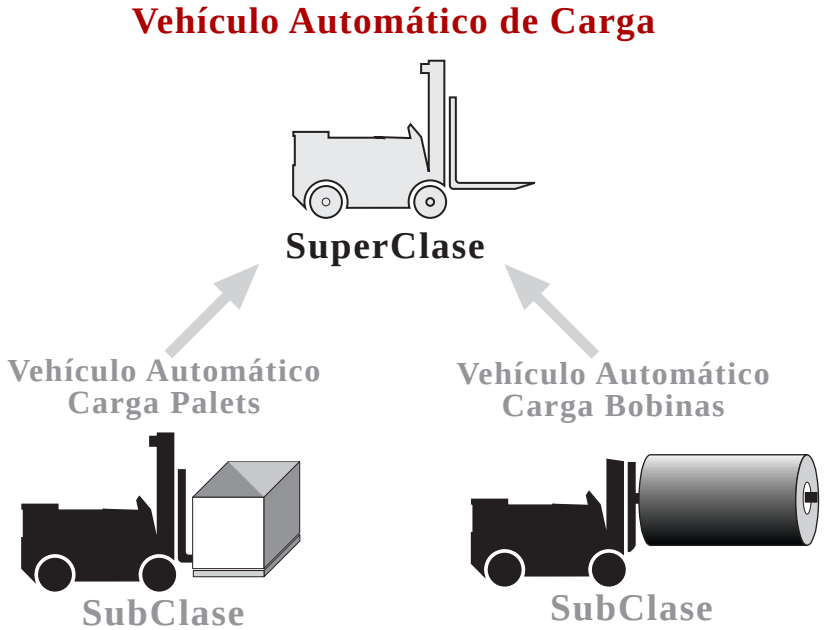
Clases

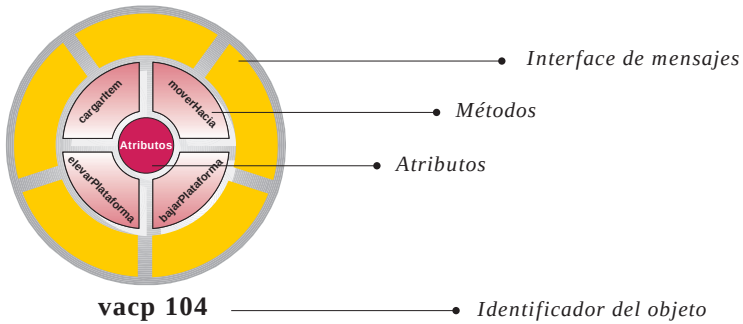
Herencia

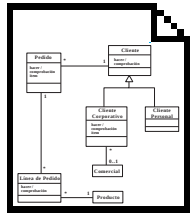
Es el mecanismo por el cual una clase de objetos puede ser definida como un caso especial de otra clase más genérica. Los casos especiales de una clase se denominan Subclases. La clase más genérica, se conoce como la Superclase de todos sus casos especiales. Además de los métodos y atributos que heredan, las Subclases definen sus propios métodos y atributos. Tambien, pueden redefinir algunos de los heredados (overriding).

La interface de mensajes definida para una Superclase también es heredada por las subclases. Esto implica que todas las Subclases de una Superclase estan preparadas para responder correctamente a los mismos mensajes que la Clase padre. Esta propiedad es extremadamente útil porque nos permite tratar de la misma manera a todas las especializaciones de una Clase.







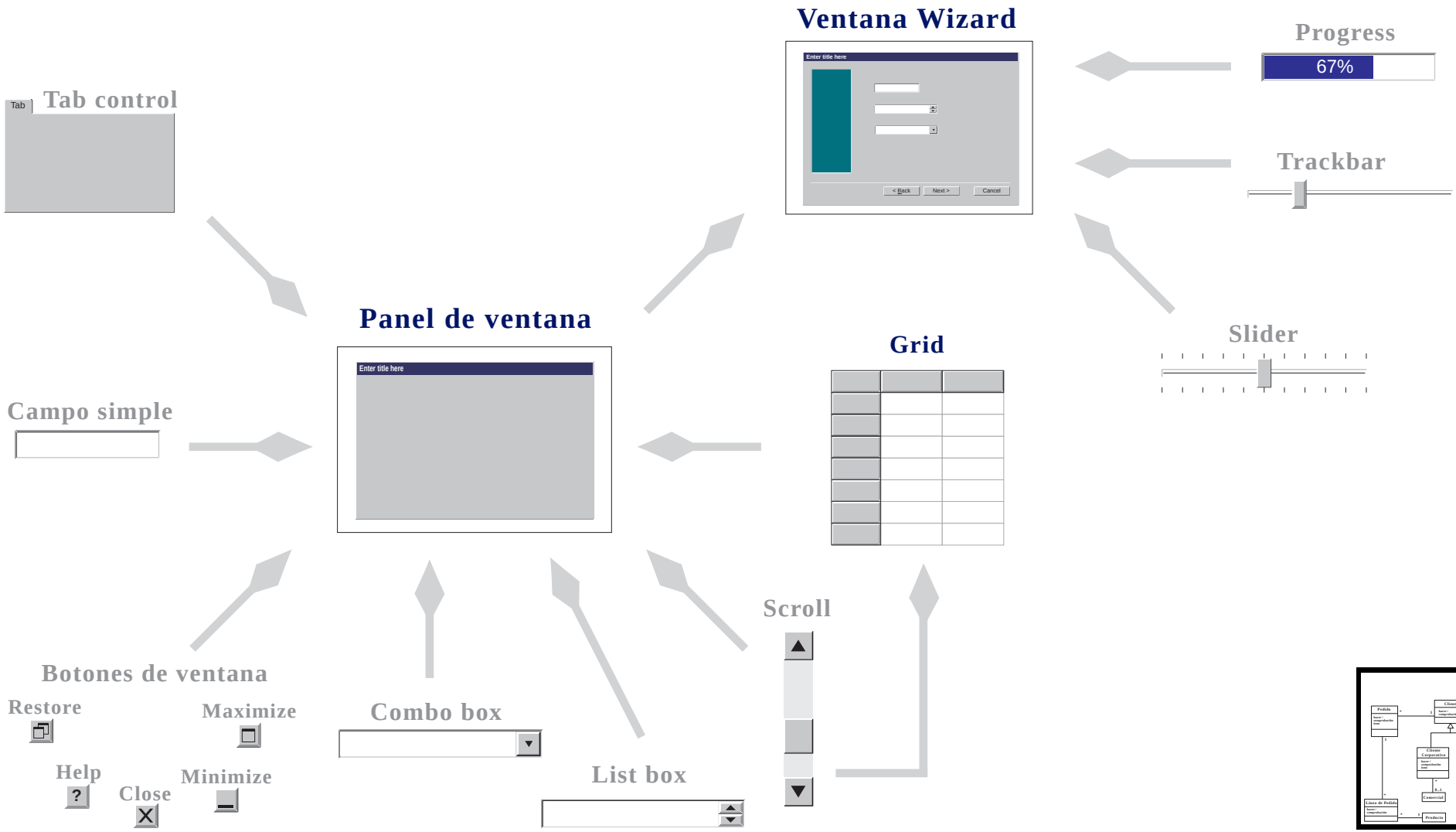


Clases

Composición

Los objetos que contienen a otros objetos se denominan Objetos Compuestos. Los atributos de un objeto pueden utilizarse de dos maneras. Podemos usarlos para almacenar valores como el número 15, o bien, el texto “Autorizado”.

También pueden contener referencias de otros objetos. Las referencias almacenadas en los atributos de un objeto compuesto, permite a este objeto enviar los mensajes apropiados a todos los objetos contenidos.



Clases

vico.org

Polimorfismo

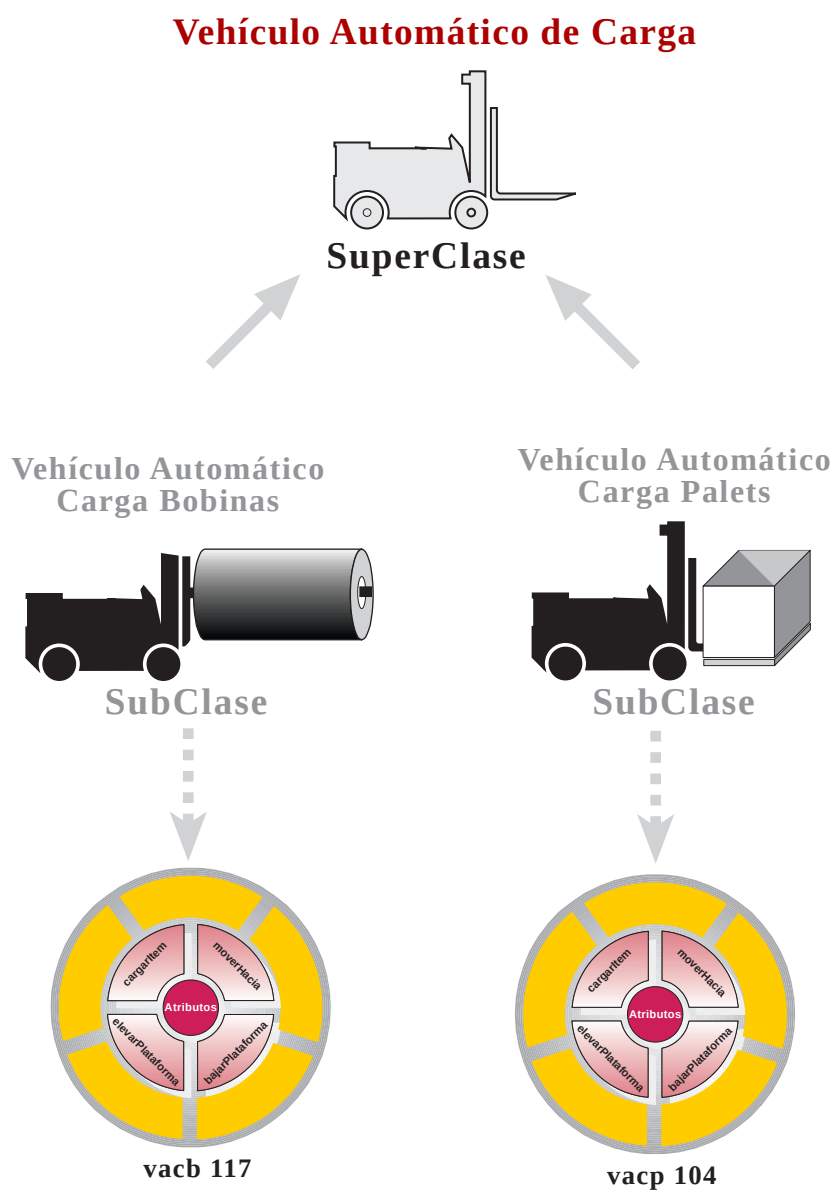
Diferentes objetos pueden responder a un mismo mensaje de diferentes maneras. El polimorfismo permite a los objetos interactuar entre ellos sin necesidad de conocer previamente a que tipo pertenecen.

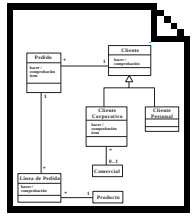
Un objeto puede ser de diferentes tipos. Por ejemplo un vehículo automático de carga puede especializarse para cargar bobinas, palets u otro tipo de items. Los demás objetos del sistema no tienen porque saber cuantos tipos de vehículos disponemos.

El mismo mensaje *cargarItem*, acompañado del parámetro que identifica dicho item, será intrepretado de distinta manera por cada objeto receptor, el cual ya conoce previamente como tiene que tratar la naturaleza de su carga: bobinas, palets, etc.

Hay una reducción de esfuerzo muy significativa por el hecho de que cualquier objeto del sistema puede enviar mensajes a los vehículos automáticos de carga sin necesidad de saber de antemano qué tipo de vehículos estan en circulación.

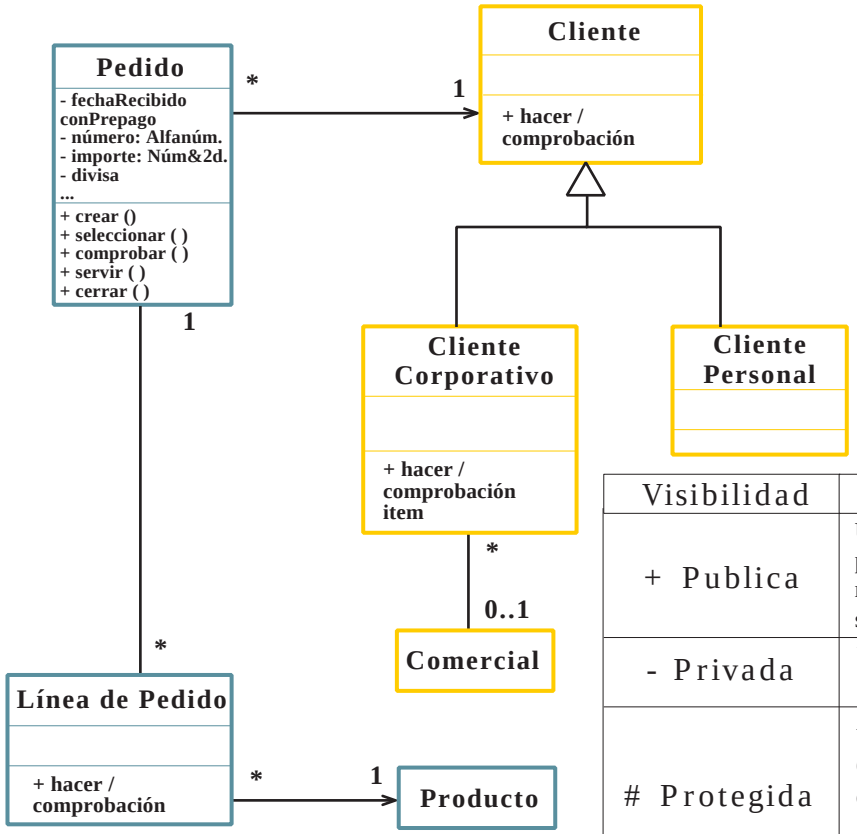
No hay necesidad así de picar un código específico para cada tipo de vehículo, con lo cual, nos ahorramos prolijas sentencias IF o CASE que son complejas de mantener y de actualizar cuando se incorporan nuevos tipos de objetos.





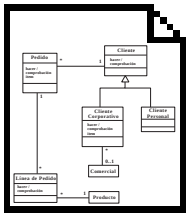
Clases

Visibilidad



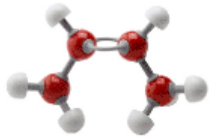
- Toda Clase encapsula unos elementos (atributos y operaciones) que disponen de ciertos criterios de visibilidad y manipulación para otras Clases.
- Los elementos públicos pueden ser usados por cualquier otra Clase.
- Los elementos privados pueden ser usados sólo por la Clase propietaria.
- Cada plataforma de desarrollo (C++, Smalltalk, Java) desarrolla sus propias reglas con respecto a la visibilidad y manipulación de atributos y operaciones.
- La notación UML especifica que todo atributo y operación de una Clase ha de disponer de un indicador de visibilidad.

Visibilidad	C++	Smalltalk	Java
+ Publica	Un elemento siempre es visible en cualquier parte del programa y puede ser llamado y modificado por cualquier objeto del sistema.	Todas las operaciones son públicas por defecto.	Un elemento siempre es visible en cualquier parte del programa y puede ser llamado y modificado por cualquier objeto del sistema.
- Privada	Un elemento sólo puede ser usado por la Clase que lo define.	Todas las variables instanciadas son privadas.	Un elemento sólo puede ser usado por la Clase que lo define.
# Protegida	Un elemento sólo puede ser usado por la Clase que lo define, o por las subclases de dicha Clase.		Un elemento puede ser usado por subclases y también por cualquier otra Clase en el mismo Package como la Clase propietaria. Esto implica que en Java, el concepto de visibilidad protegida es más público que package.
Package			Un elemento sólo puede ser usado por otras Clases que compartan el mismo Package.
Ejemplos	Consideremos la Clase **CLIENTE** que dispone de una subclase **CLIENTE PERSONAL**. Consideremos también que el objeto <<**Josep V.**>>, es una instancia de **CLIENTE PERSONAL**. <<**Josep V.**>> • Puede usar todo elemento público de cualquier objeto del sistema. • Puede usar también todo elemento privado de la Clase **CLIENTE PERSONAL**. • No puede usar los elementos privados definidos en la Clase **CLIENTE**. • Puede usar los elementos protegidos definidos para **CLIENTE** y **CLIENTE PERSONAL**. En C++, podemos acceder a elementos de otros objetos de la propia Clase, de la misma manera que podemos acceder a los propios elementos de un objeto.	<<**Josep V.**>>, puede acceder a cualquier variable instanciada dentro de su propio objeto, si dicha variable ha sido definida dentro de **CLIENTE** o **CLIENTE PERSONAL**. De esta manera, el concepto de visibilidad privada en Smalltalk es parecido al concepto de visibilidad protegida en C++. En Smalltalk no hay diferencia con respecto a que un objeto sea de la misma Clase o no. Podemos acceder sólo a los elementos públicos de otro objeto. En Smalltalk, <<**Miquel M.**>>, no puede acceder a las variables instanciadas privadas de <<**Josep V.**>>, sólo a sus operaciones públicas.	Java permite marcar también las Clases como públicas o packages. Los elementos de una Clase pública pueden ser usados por cualquier Clase que importe el package a la que pertenece la Clase. Los elementos de una Clase package sólo pueden ser usados por otras Clases del mismo package.



Clases

Consideremos la instancia de **CLIENTE PERSONAL**, <<**Miquel M.**>>, este objeto puede acceder a cualquier elemento de <<**Josep V.**>>, que ha sido definido también como una instancia de **CLIENTE PERSONAL**, sea público, privado o protegido. <<**Miquel M.**>>, a su vez también puede acceder a cualquier elemento privado, protegido o público de <<**Josep V.**>>



vico.org

Descripción de la Dinámica del Sistema

La dinámica de un sistema está determinada por:

- Todos los posibles estados que sus objetos pueden tener.
- Todas las posibles secuencias de eventos que pueden ocurrir.
- Todas las transiciones posibles, de un estado a otro, como consecuencia de los eventos que afectan a los objetos.

CU Realizar pedido

Análisis textual del Use Case

Flujo Principal	Variaciones
1. Usuario *identifica el cliente* que ha enviado un pedido.	
2. Usuario *entra lineas de pedido*, con su código de artículo, tipo de presentación y cantidad.	
3. Sistema *comprueba cada línea* del pedido para validar la situación del artículo en catálogo y el número de items del artículo en stock.	**a.** Artículo NO está vigente en catálogo, sistema informa que *articulo no está vigente* y muestra artículos alternativos activando el **CU Seleccionar artículo**. **b.** SI existen suficientes items del artículo en el stock, sistema *asigna items al pedido*. **c.** NO existen suficientes items del artículo en stock, o la asignación de items deja la situación del artículo en stock por debajo del nivel de reposición, sistema *genera pedido de reposición* activando el **CU Generar pedido**.
4. Sistema *comprueba la situación del pedido* .	**a.** SI se ha realizado el pago y SI todos los items del pedido han sido asignados, sistema informa que procede a *servir el pedido* activando el **CU Servir pedido**. **b.** SI se ha realizado el pago y NO existen suficientes items del artículo en stock, sistema *aparca el pedido* del cliente activando el **CU Aparcar pedido**. **c.** SI no se ha realizado el pago según el plazo convenido, sistema *cancela el pedido*.

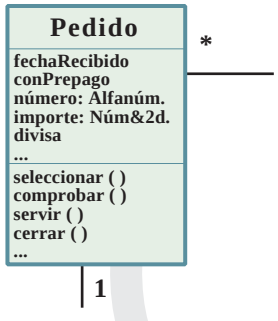
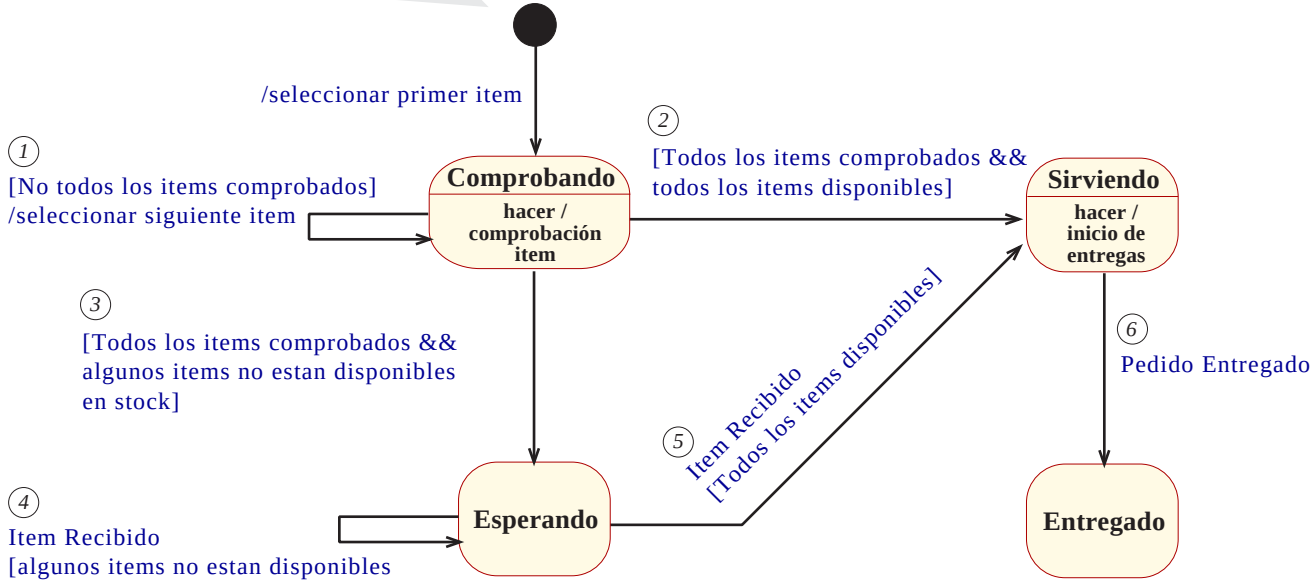
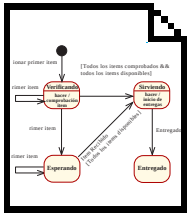


Diagrama de Estados de un Pedido

Notación UML 1.3





Dinámica Estados

Descripción de la Dinámica del Sistema

Utilizamos el diagrama de estados para describir el comportamiento de una Clase dentro de una serie temporal.

CU Realizar pedido

Flujo Principal	Variaciones
1. Usuario *identifica el cliente* que ha enviado un pedido.	
2. Usuario *entra lineas de pedido*, con su código de artículo, tipo de presentación y cantidad.	
3. Sistema *comprueba cada línea* del pedido para validar la situación del artículo en catálogo y el número de items del artículo en stock.	**a.** Artículo NO está vigente en catálogo, sistema informa que *articulo no está vigente* y muestra artículos alternativos activando el **CU Seleccionar artículo**. **b.** SI existen suficientes items del artículo en el stock, sistema *asigna items al pedido*. **c.** NO existen suficientes items del artículo en stock, o la asignación de items deja la situación del artículo en stock por debajo del nivel de reposición, sistema *genera pedido de reposición* activando el **CU Generar pedido**.

Análisis textual del Use Case

Clase

Pedido

Atributos: fechaRecibido, conPrepago, número: Alfanúm., importe: Núm&2d., divisa, ...

Operaciones: seleccionar ( ), comprobar ( ), servir ( ), cerrar ( ), ...

* — 1

Sintaxis para etiquetar una transición

Evento [Indicador] / Acción

Los tres elementos son opcionales

/ Acción — *Transición*

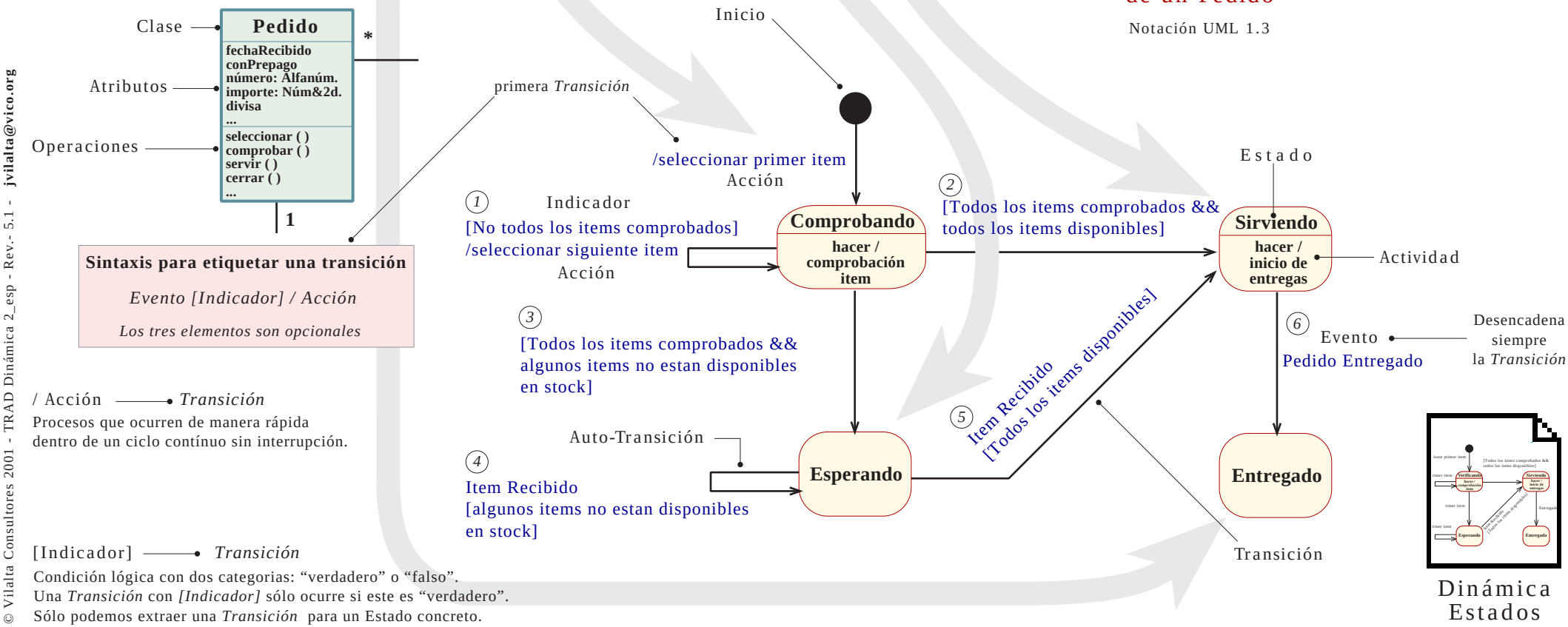
Procesos que ocurren de manera rápida dentro de un ciclo contínuo sin interrupción.

[Indicador] — *Transición*

Condición lógica con dos categorias: “verdadero” o “falso”.
Una *Transición* con *[Indicador]* sólo ocurre si este es “verdadero”.
Sólo podemos extraer una *Transición* para un Estado concreto.

Diagrama de Estados de un Pedido

Notación UML 1.3



Dinámica Estados

Descripción de la Dinámica del Sistema

Construimos el diagrama de estados a partir de una clase concreta para mostrar el comportamiento de un objeto durante su ciclo de vida.

CU Realizar pedido

Flujo Principal	Variaciones
1. Usuario *identifica el cliente* que ha enviado un pedido.	
2. Usuario *entra lineas de pedido*, con su código de artículo, tipo de presentación y cantidad.	
3. Sistema *comprueba cada línea* del pedido para validar la situación del artículo en catálogo y el número de items del artículo en stock.	**a.** Artículo NO está vigente en catálogo, sistema informa que *articulo no está vigente* y muestra artículos alternativos activando el **CU Seleccionar artículo**. **b.** SI existen suficientes items del artículo en el stock, sistema *asigna items al pedido*. **c.** NO existen suficientes items del artículo en stock, o la asignación de items deja la situación del artículo en stock por debajo del nivel de reposición, sistema *genera pedido de reposición* activando el **CU Generar pedido**.

Análisis textual del Use Case

Diagrama de Estados de un Pedido

Notación UML 1.3

Clase

Pedido

Atributos: **fechaRecibido**, **conPrepago**, **número: Alfanúm.**, **importe: Núm&2d.**, **divisa**, **...**

Operaciones: **seleccionar ( )**, **comprobar ( )**, **servir ( )**, **cerrar ( )**, **...**

* — 1

Implementación

Sintaxis para etiquetar una transición

Evento [Indicador] / Acción

Los tres elementos son opcionales

Inicio

primera *Transición*

/seleccionar primer item
Acción

Comprobando
hacer / comprobación item

1 Indicador
[No todos los items comprobados]
/seleccionar siguiente item
Acción

2 [Todos los items comprobados && todos los items disponibles]

Sirviendo
hacer / inicio de entregas

Estado

Actividad

Cuando una *Transición* no dispone de un *Evento* en su etiqueta, significa que la *Transición* se realiza tan pronto como la *Actividad* asociada con un *Estado* es finalizada

Aunque finalize la *Actividad* el Pedido permanecerá en este *Estado* hasta que ocurre el *Evento* que desencadena su *Transición*

3 [Todos los items comprobados && algunos items no estan disponibles en stock]

4 Item Recibido
[algunos items no estan disponibles en stock]

Auto-Transición

Esperando

5 Item Recibido
[Todos los items disponibles]

6 Evento
Pedido Entregado

Desencadena siempre la *Transición*

Entregado

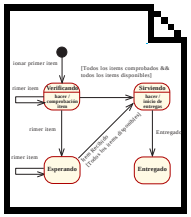
Transición

No hay *Actividades* para este *Estado*, por lo que el Pedido permanecerá a la espera de un *Evento*.

/ Acción — *Transición*
Procesos que ocurren de manera rápida dentro de un ciclo contínuo sin interrupción.

/ Actividad — *Estado*
Procesos que ocurren dentro de un ciclo discontínuo y pueden ser interrumpidospor algun evento.

[Indicador] — *Transición*
Condición lógica con dos categorias: “verdadero” o “falso”.
Una *Transición* con *[Indicador]* sólo ocurre si este es “verdadero”.
Sólo podemos extraer una *Transición* para un Estado concreto.



Dinámica Estados

Descripción de la Dinámica del Sistema

Un Evento no es un objeto. Un Evento es la "causa" que justifica la existencia de una información.

Sólo podemos conocer que un Evento ha ocurrido, detectando sus "efectos" en nuestro modelo.

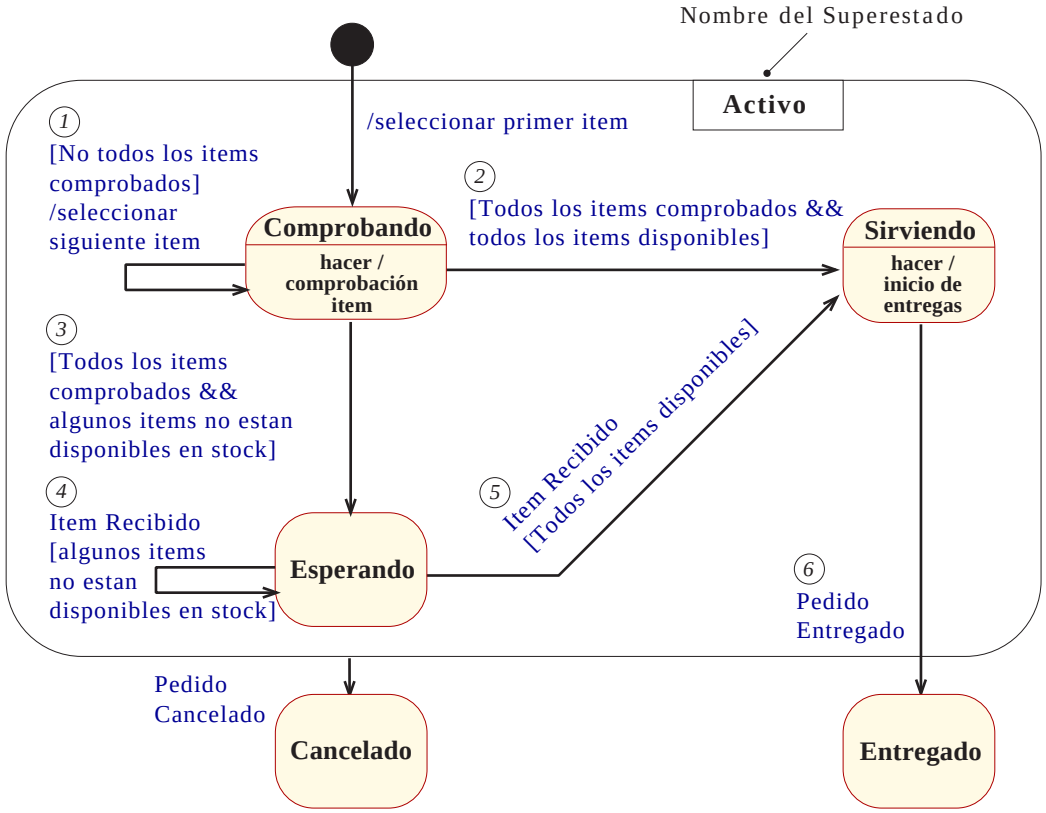
Sólo nos interesan aquellos Eventos que provocan un cambio de estado en los objetos de nuestro modelo.

Hay que distinguir un Evento como tal, del objeto que representa el registro de los efectos de dicho Evento.

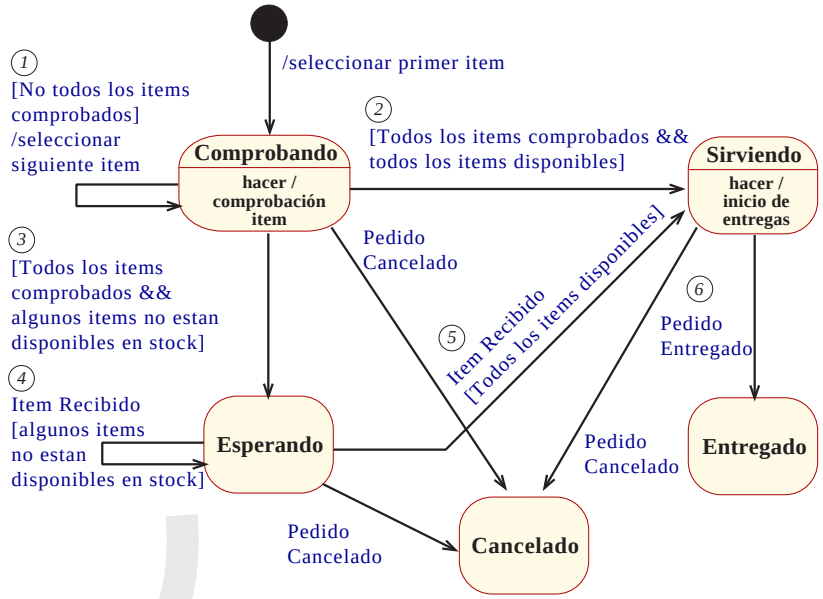
CU Realizar pedido

Análisis textual del Use Case

Flujo Principal	Variaciones
1. Usuario *identifica el cliente* que ha enviado un pedido.	
2. Usuario *entra lineas de pedido*, con su código de artículo, tipo de presentación y cantidad.	
3. Sistema *comprueba cada línea* del pedido para validar la situación del artículo en catálogo y el número de items del artículo en stock.	**a.** Artículo NO está vigente en catálogo, sistema informa que *articulo no está vigente* y muestra artículos alternativos activando el **CU Seleccionar artículo**. **b.** SI existen suficientes items del artículo en el stock, sistema *asigna items al pedido*. **c.** NO existen suficientes items del artículo en stock, o la asignación de items deja la situación del artículo en stock por debajo del nivel de reposición, sistema *genera pedido de reposición* activando el **CU Generar pedido**.



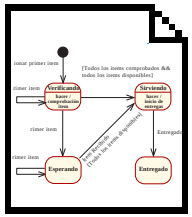
con Superestados



sin Superestados

Diagrama de Estados

Notación UML 1.3



Dinámica Estados

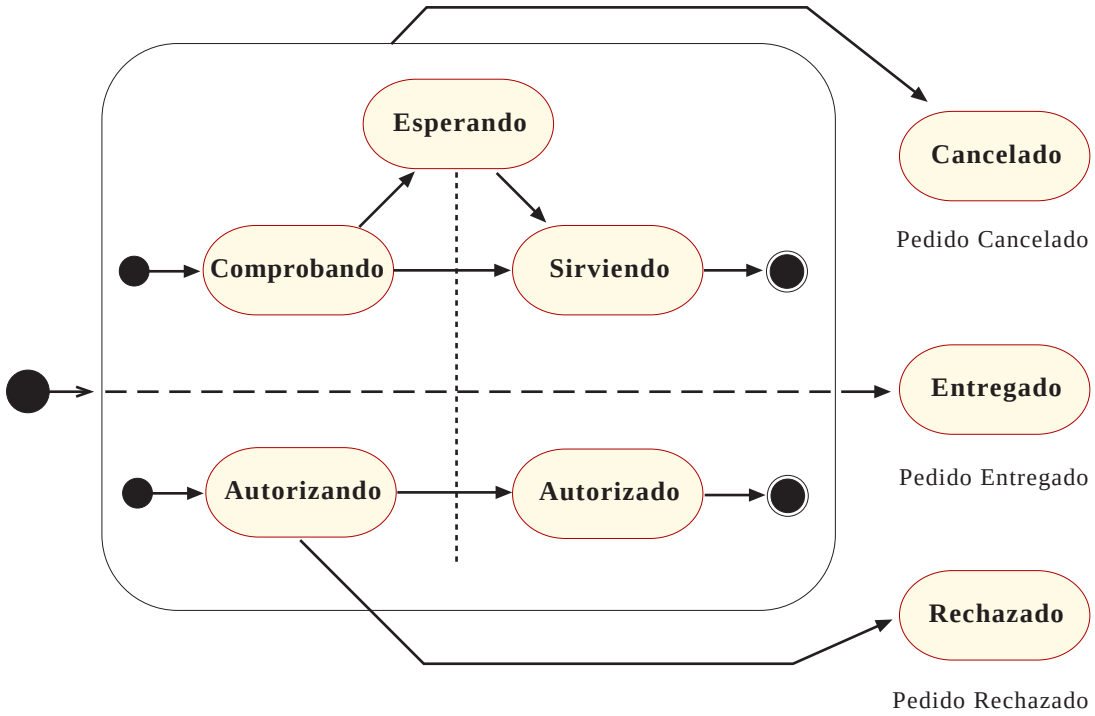
Descripción de la Dinámica del Sistema

Los diagramas de estados son muy útiles para describir el comportamiento de un objeto a través de múltiples Casos de Uso.

CU Realizar pedido

Flujo Principal	Variaciones
1. Usuario *identifica el cliente* que ha enviado un pedido.	
2. Usuario *entra lineas de pedido*, con su código de artículo, tipo de presentación y cantidad.	
3. Sistema *comprueba cada línea* del pedido para validar la situación del artículo...	**a.** Artículo NO está vigente en catálogo, sistema informa que *articulo no está vigente* y muestra... **b.** SI existen suficientes items... **c.** NO existen suficientes items...
4. Sistema *comprueba la situación del pedido* .	**a.** SI se ha realizado el pago y SI todos los items del pedido han sido asignados, sistema informa que procede a *servir el pedido* activando el **CU Servir pedido**. **b.** SI se ha realizado el pago y NO existen suficientes items del artículo en stock, sistema *aparca el pedido* del cliente activando el **CU Aparcar pedido**. **c.** SI no se ha realizado el pago según el plazo convenido, sistema *cancela el pedido*.

Análisis textual del Use Case



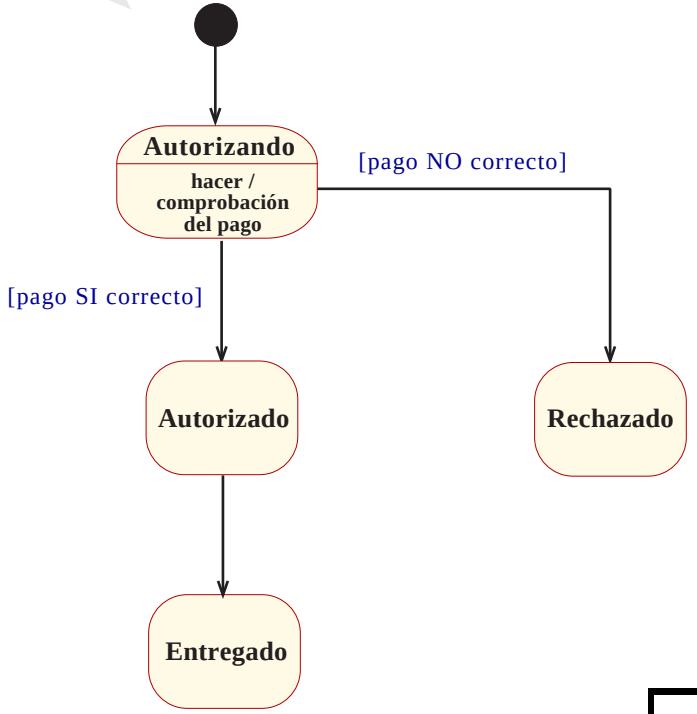
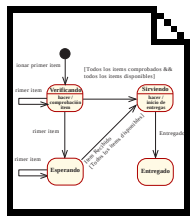


Diagrama de Estados Concurrentes

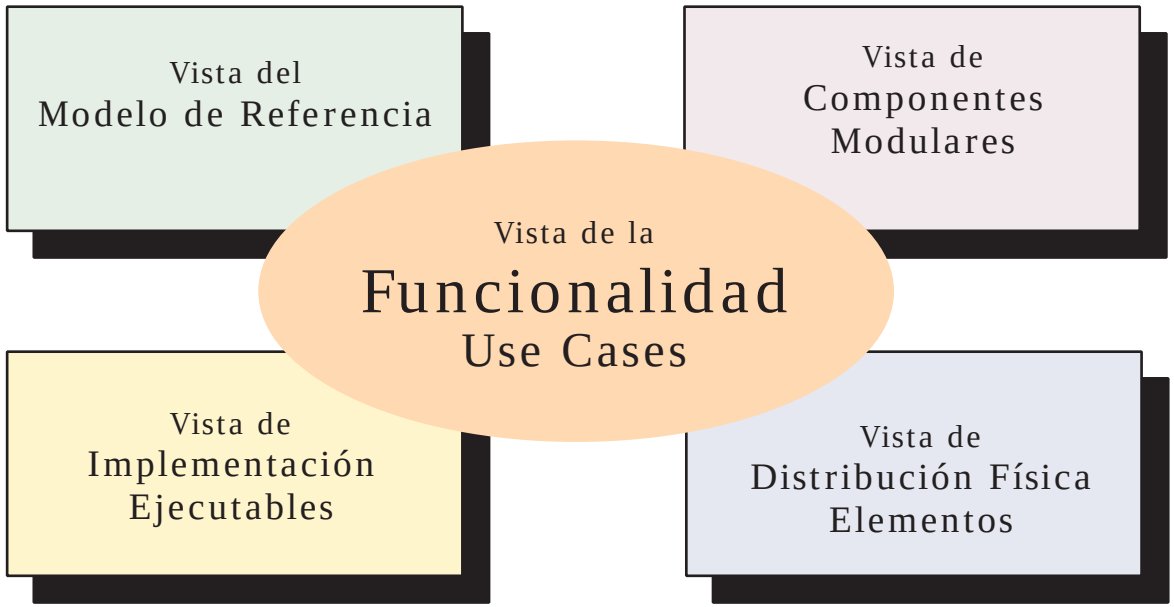
Notación UML 1.3



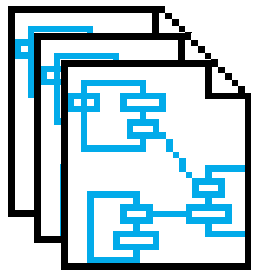
Dinámica Estados

Arquitectura del sistema

Una Arquitectura es un conjunto organizado de elementos. Se utiliza para especificar las decisiones estratégicas acerca de la estructura y funcionalidad del sistema, las colaboraciones entre sus distintos elementos y su despliegue físico para cumplir unas responsabilidades bien definidas.

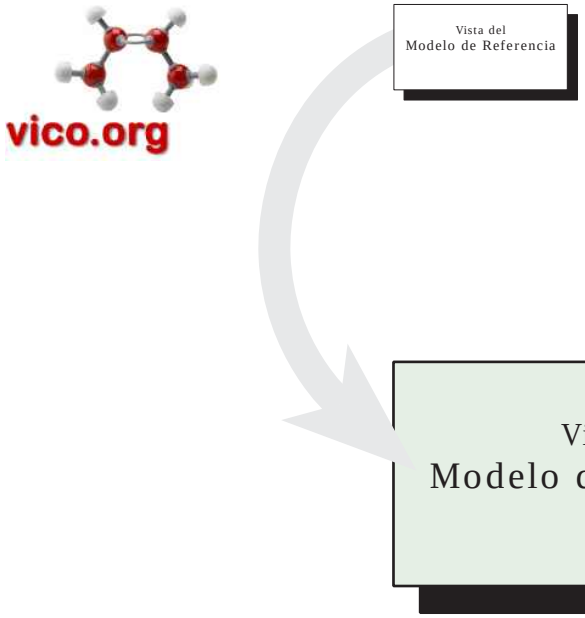


Vista de la Arquitectura 4 + 1
Notación UML 1.3



Arquitectura

Arquitectura del sistema



La vista del Modelo de Referencia (Reference Information Model), está determinada por la arquitectura lógica.

La arquitectura lógica es capturada por los diagramas de Clases que contienen las Clases y relaciones que representan las abstracciones esenciales del sistema a desarrollar.

- Clases
- Asociaciones
- Agregaciones
- Generalización
- Packages

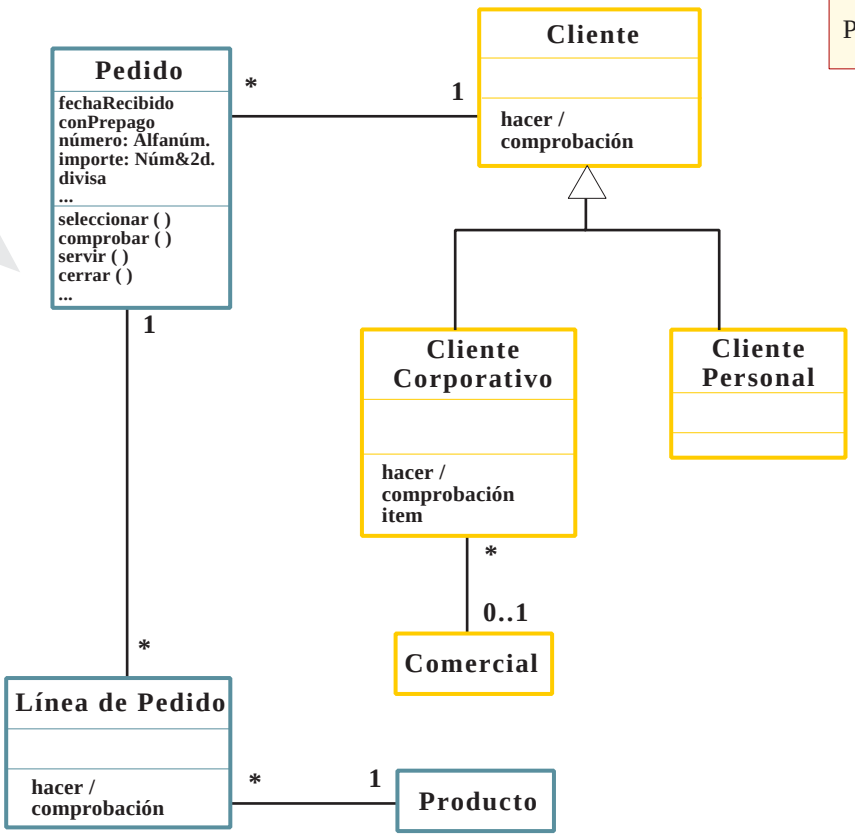
El Modelo de Referencia se construye en sucesivas iteraciones durante la fase de Formalización.

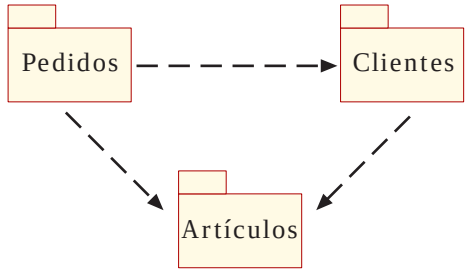
Las Clases y Packages del modelo reflejan las decisiones tomadas con respecto a los “mecanismos clave” del sistema.

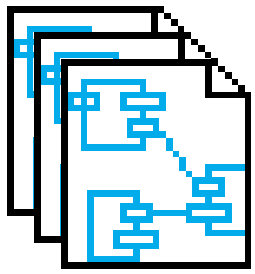
Una eficiente implementación de los “mecanismos clave” requiere seleccionar Patrones (Patterns) que se ajusten a los requerimientos esenciales del proyecto.

La implementación de “mecanismos clave” requiere seleccionar también:

- Lenguaje de programación
- Motor de Base de Datos
- Interface gráfico de usuario “look and feel”
- Tratamiento de errores
- Mecanismos de comunicación
- Migración y distribución de objetos
- Networking

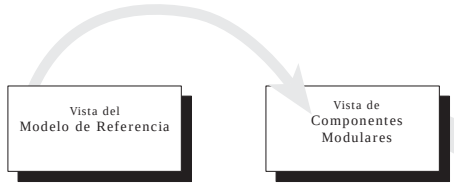






Arquitectura

Arquitectura del sistema



La vista de Componentes Modulares refleja la organización de módulos de software dentro del entorno de desarrollo.

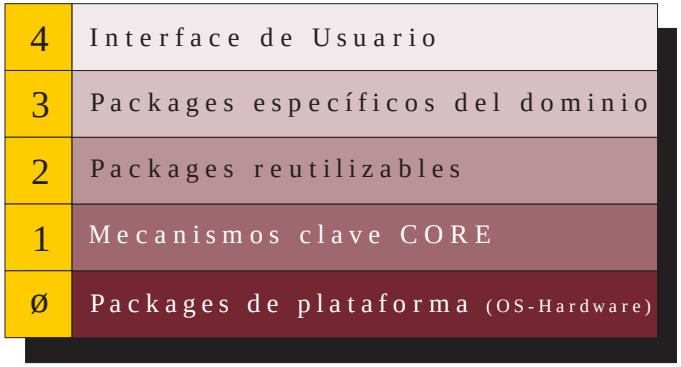
Esta vista de arquitectura toma en cuenta los requerimientos que facilitan la programación, los niveles de reutilización, y las limitaciones impuestas por el entorno de desarrollo.

Disponemos de dos elementos para modelizar esta vista:

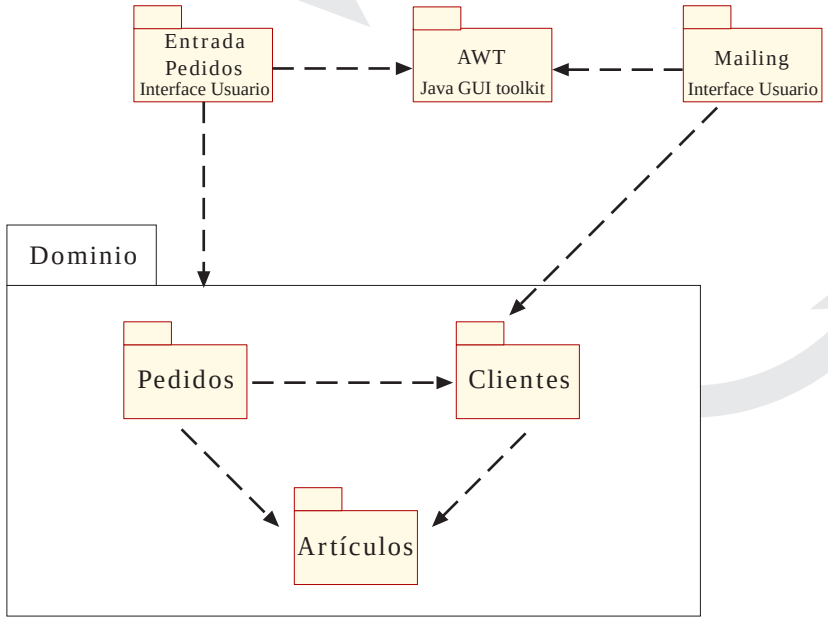
- Packages: En esta vista representan una partición física del sistema.
- Componentes: En esta vista representan la organización de módulos de código fuente.

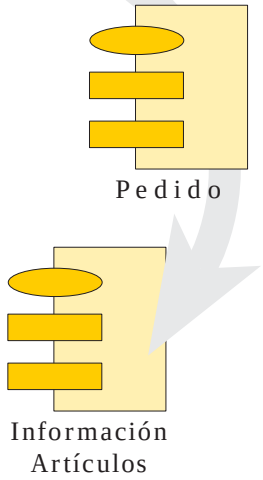
Vista de
Componentes
Modulares

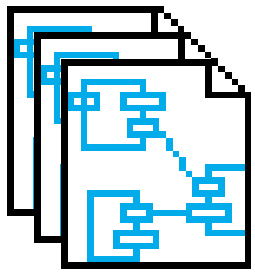
Los Packages estan organizados en una jerarquía de capas que disponen de una interface bien definida:



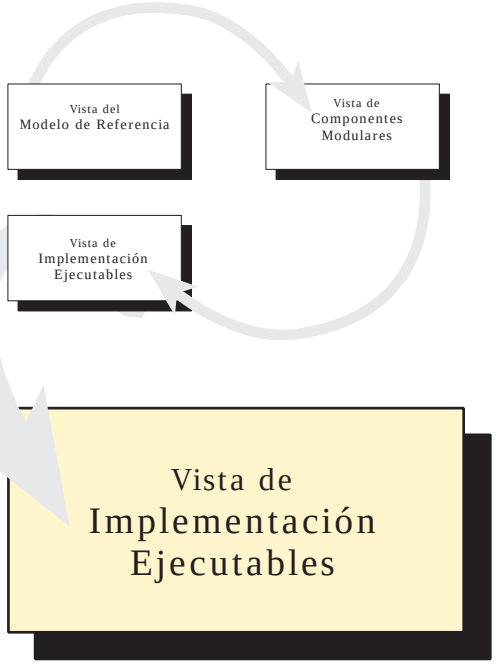
Los Packages de la vista lógica del modelo estan mapeados con los Packages físicos y los componentes de software (subsistemas).







Arquitectura del sistema



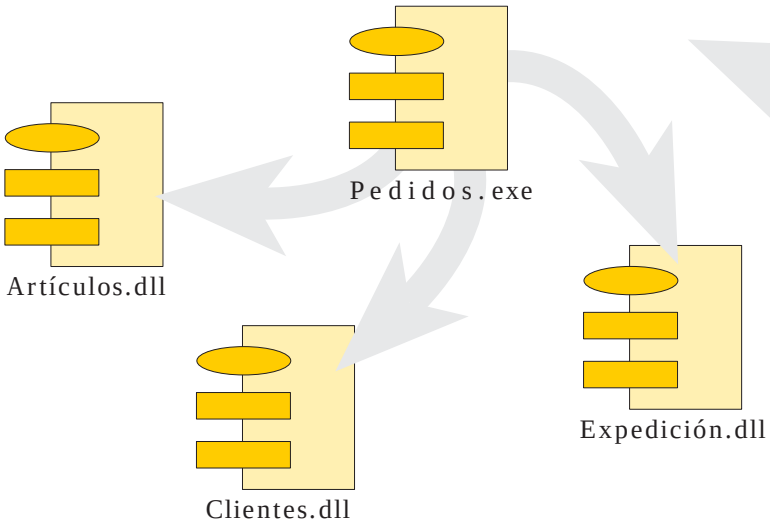
Esta vista se centra en la estructura de los componentes "run-time", los ejecutables del sistema.

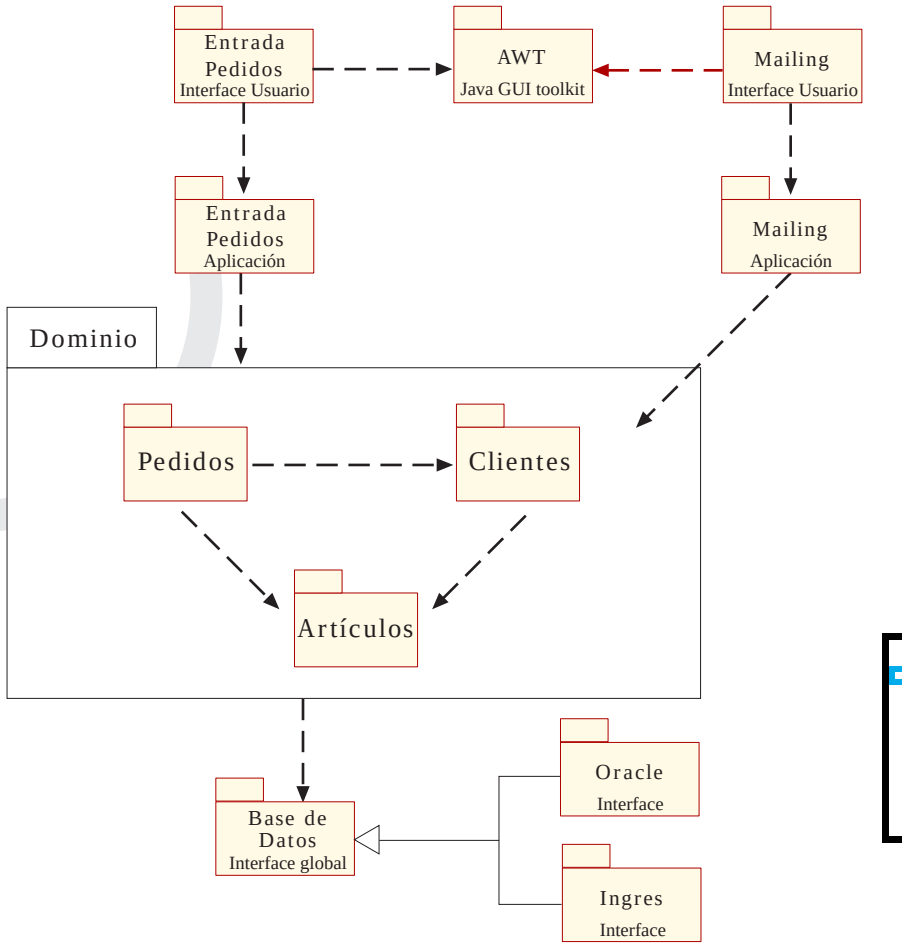
Esta arquitectura tiene muy en cuenta los siguientes requerimientos no funcionales:

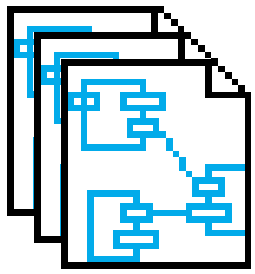
- Rendimiento
- Fiabilidad
- Escalabilidad
- Integridad
- Seguridad
- Sincronización
- Administración del sistema

Los componentes run-time muestran los mappings de las Clases a librerías de tipo ActiveX, Applets de Java y librerías dinámicas.

Los componentes ejecutables muestran sus interfaces y niveles de dependencia dentro de la aplicación.







Arquitectura

Arquitectura del sistema

Esta vista presenta el mapping de componentes de software ejecutables con los nodos de procesamiento (hardware).

Esta arquitectura tiene en cuenta los siguientes requerimientos:

- Disponibilidad del sistema
- Rendimiento
- Escalabilidad

Los diagramas de distribución muestran el despliegue de nodos (locales y remotos), en la organización de la empresa.

Permite al equipo de desarrollo comprender mejor la topología de un sistema distribuido.

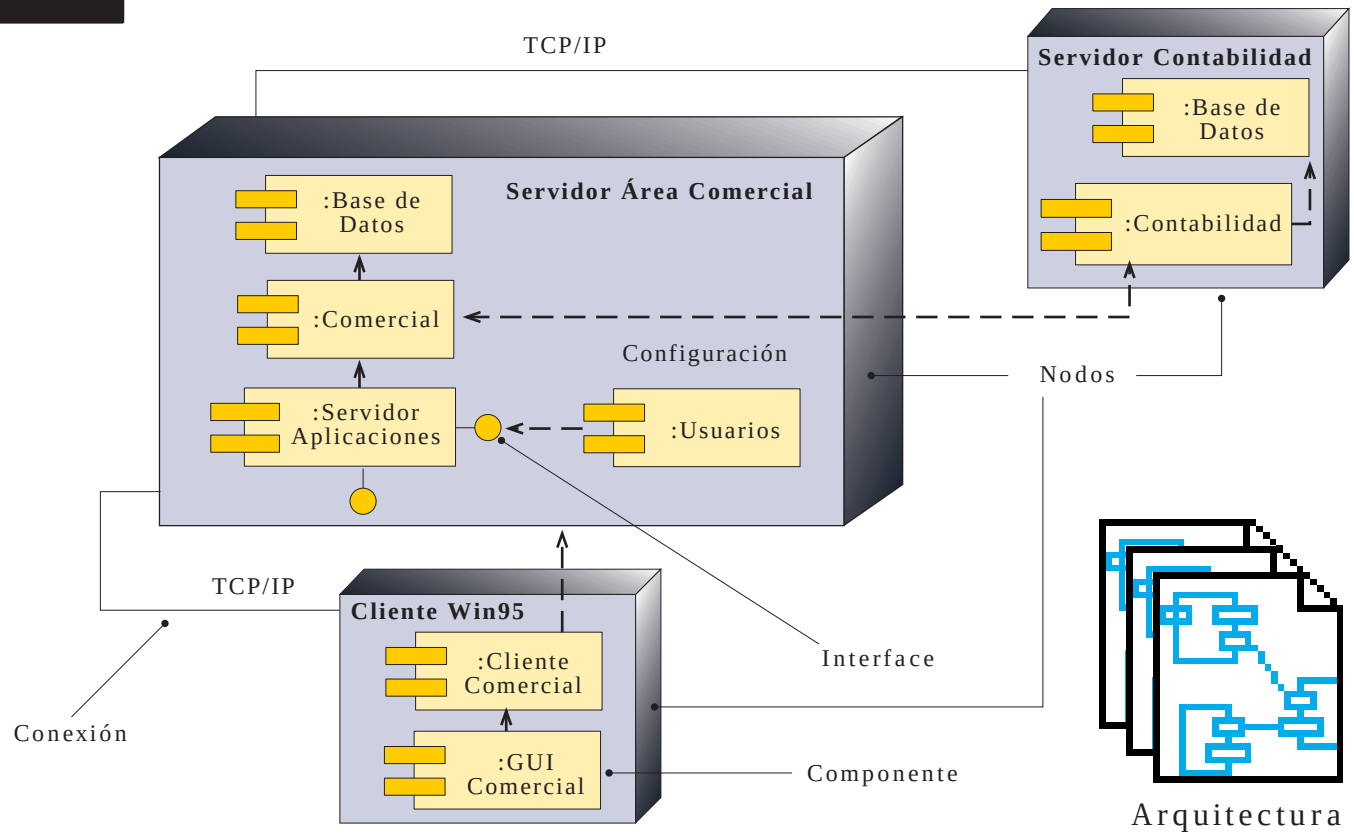
Un Nodo es un objeto físico run-time que representa un recurso informático. Este recurso, generalmente dispone de datos persistentes y capacidad de proceso.

En la mayoría de los casos un Nodo puede representar una pieza de hardware, desde un periférico a un servidor.

Las Conexiones entre Nodos muestran las líneas de comunicación con las que el sistema tendrá que interactuar.

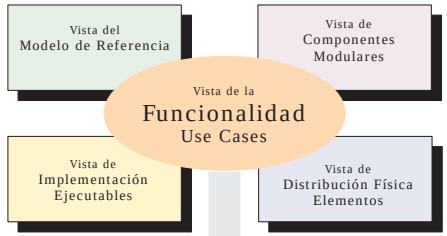
Los Componentes, en un diagrama de distribución, representan los módulos físicos de código, los cuales, se corresponden con los Packages de ejecutables. De esta manera, el diagrama muestra donde corre cada Package en el sistema.

Las Dependencias muestran cómo los componentes se comunican con otros componentes. La dirección de una Dependencia concreta, indica el conocimiento en la comunicación.



Arquitectura

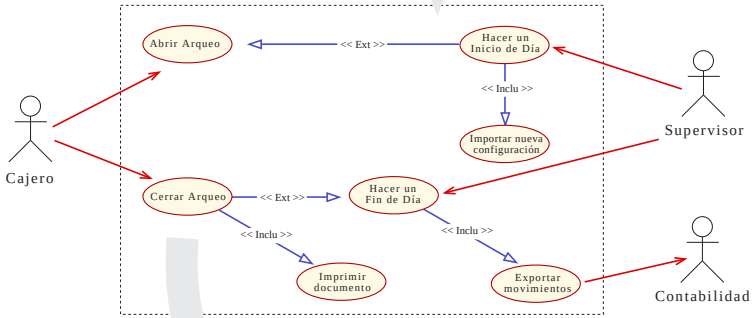
Arquitectura del sistema



Esta vista certifica la validez de:

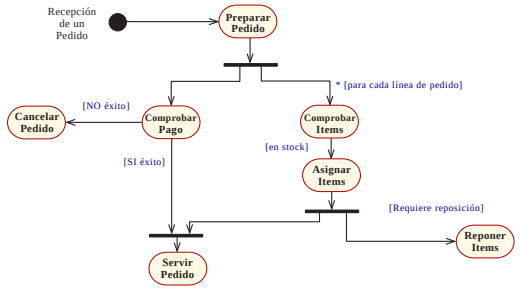
- Modelo de Referencia
- Componentes modulares de software
- Ejecutables
- Distribución de recursos informáticos

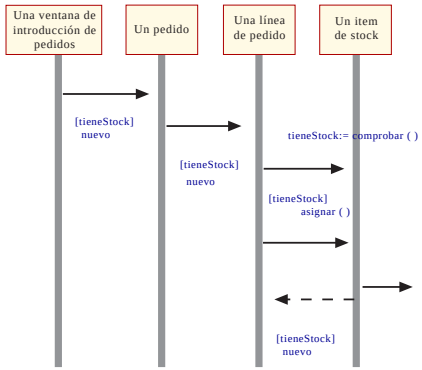
Con la funcionalidad requerida del sistema.



Utilizamos los siguientes elementos para describir la funcionalidad:

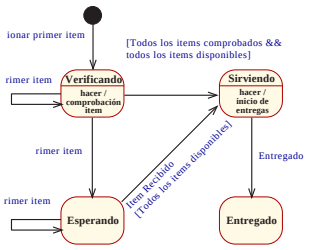
- Diagramas de Casos de Uso
- Especificación de Casos de Uso
- Diagramas de Interacción (Escenarios)
- Diagramas de Actividad (Flujos de Trabajo)
- Diagramas de Estados (Dinámica)

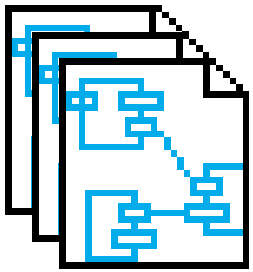




CU Realizar pedido

Flujo Principal	Variaciones
1. Usuario *identifica el cliente* que ha enviado un pedido.	
2. Usuario *entra lineas de pedido*, con su código de artículo, tipo de presentación y cantidad.	
3. Sistema *comprueba cada línea* del pedido para validar la situación del artículo en catálogo y el número de items del artículo en stock.	**a.** Artículo NO está vigente en catálogo, sistema informa que *articulo no está vigente* y muestra artículos alternativos activando el **CU Seleccionar artículo**. **b.** SI existen suficientes items del artículo en el stock, sistema *asigna items al pedido*. **c.** NO existen suficientes items del artículo en stock, o la asignación de items deja la situación del artículo en stock por debajo del nivel de reposición, sistema *genera pedido de reposición* activando el **CU Generar pedido**.





Arquitectura