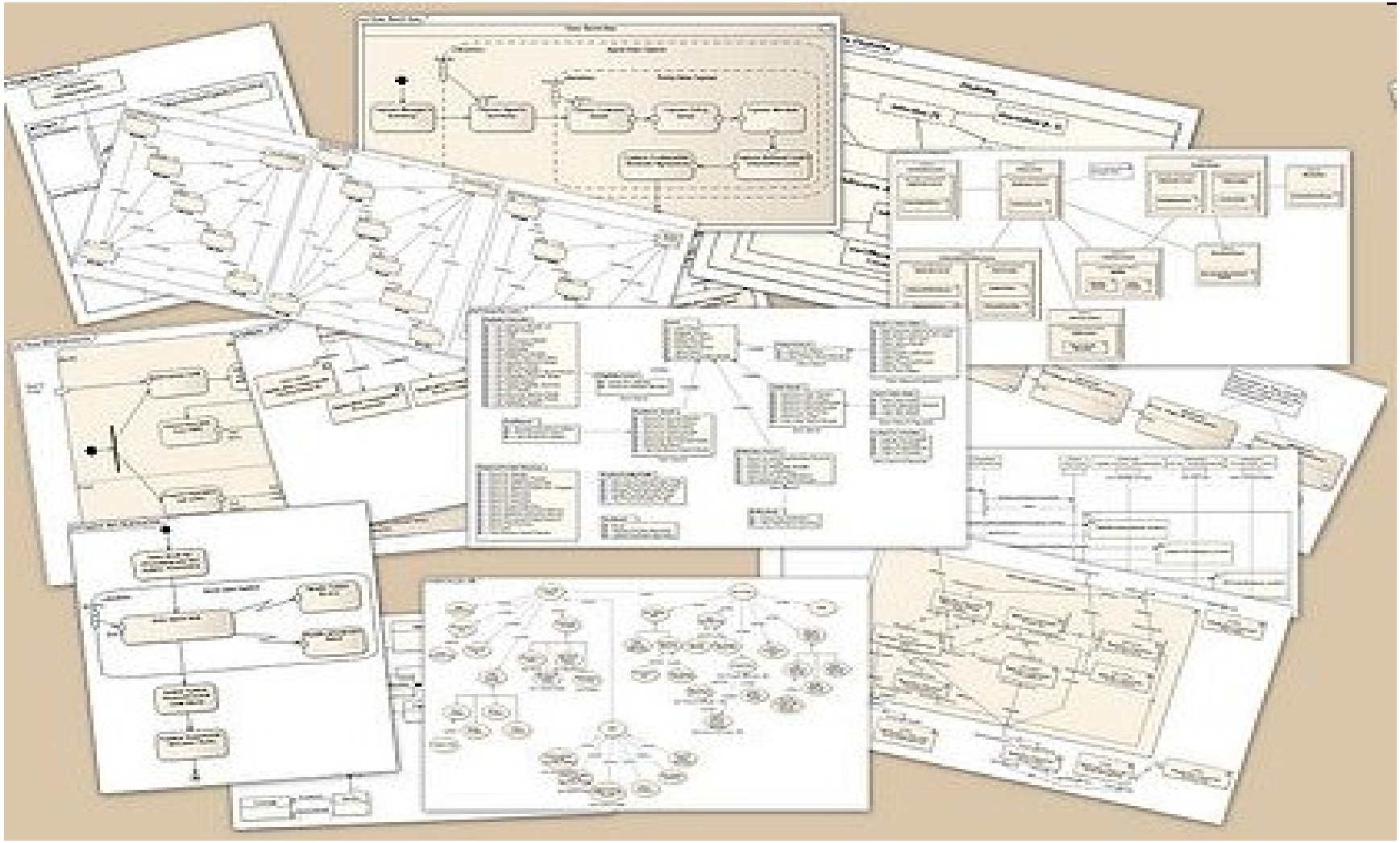
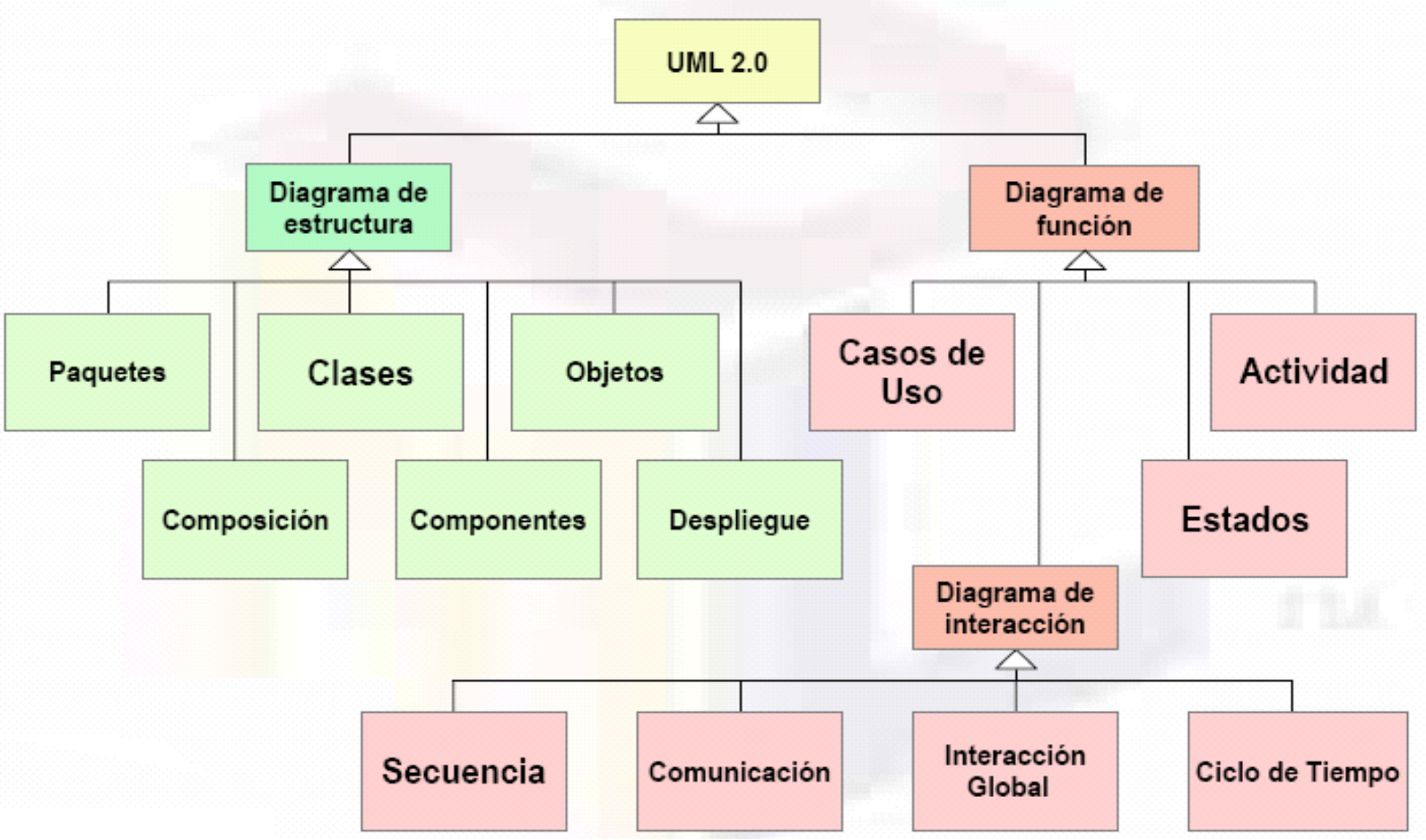
UML

- Unified Modeling Language
- Objetivo: Proveer un lenguaje común que puede ser usado para el desarrollo de software
- Lenguaje que permite:
  - Visualizar: La comunicación es a través de gráficos
  - Especificar: construyendo modelos para el análisis, diseño, implementación
  - Construir: Permite la generación de código a partir de un modelo UML, y la construcción de un modelo a partir del código (ingeniería reversa)
  - Documentar: Permite la documentación completa de todo el sistema
- Aprobado como estándar por la OMG en 1997
- Actualmente se encuentra en la versión 2.1.2 (nov 2007)

Diagramas en UML



Diagramas en UML



Tipos de Diagramas

- **Modelo Estático**
  - Construye y documenta los aspectos estáticos de un sistema.
  - Refleja la estructura básica y estable de un sistema software.
  - Crea una representación de los principales elementos del dominio del problema

- **Modelo Dinámico**
  - Crea los diagramas que muestran el comportamiento de un sistema
- Para requisitos se utilizan los siguientes diagramas:
  - Diagrama de Casos de Uso
  - Diagrama de Clases (Modelo Conceptual)
  - Diagrama de Actividad
  - Diagramas de Máquinas de Estado

Diagrama de Casos de Uso

- Permite visualizar en una forma compacta los casos de uso del sistema y que actores participan en cada caso de uso
- Presenta las relaciones que existen entre los casos de uso
- Muestra los límites del sistema
- Visión estática de los Casos de Uso de un sistema
- Consta de los siguientes elementos:
  - Actor
  - Caso de Uso
  - Relaciones
    - Include
    - Extend
    - Generalización

Diagrama de Casos de Uso - Ejemplo

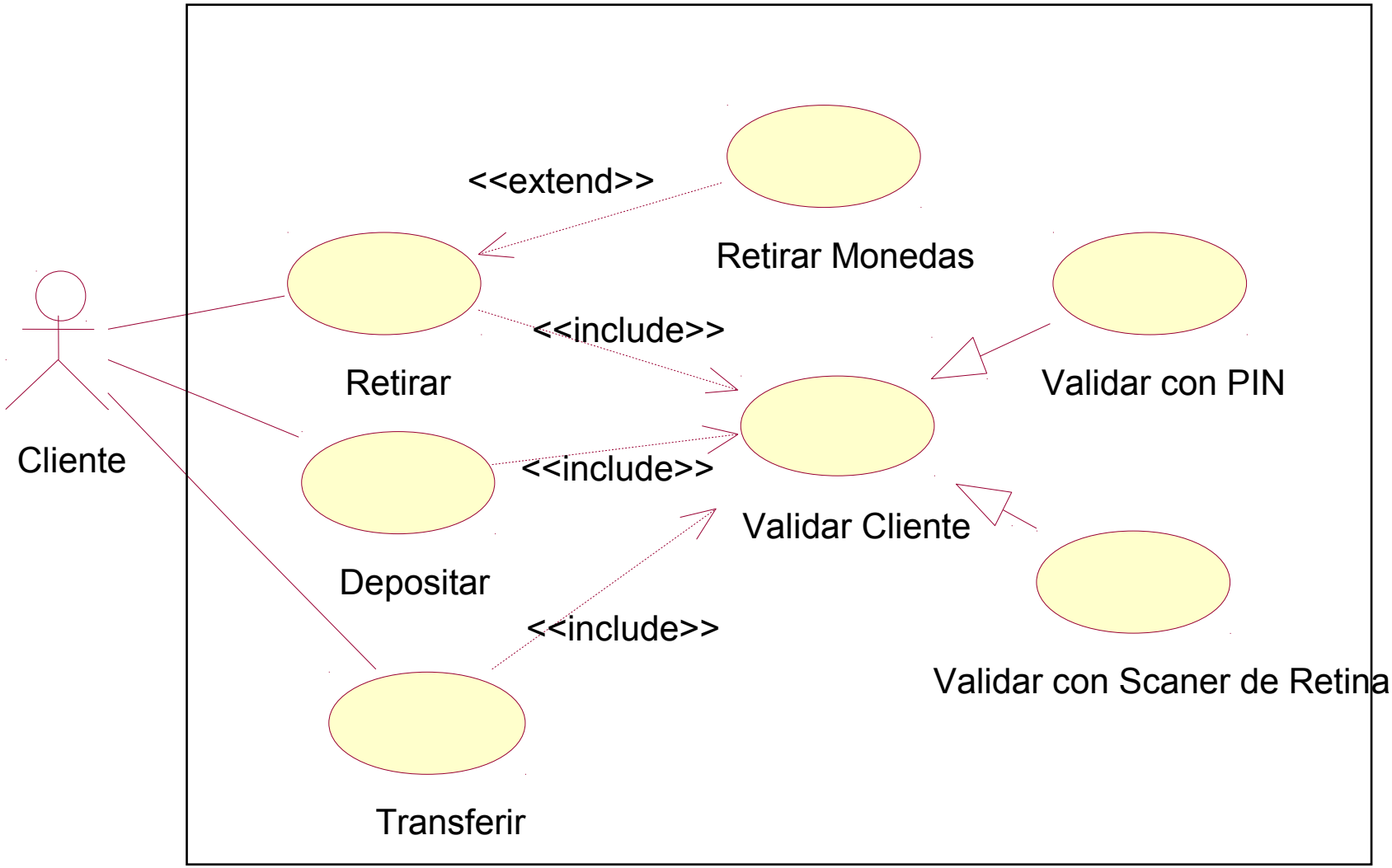
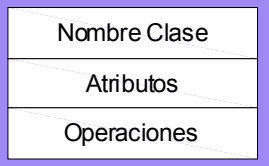


Diagrama de Clases



- Muestra las clases e interfaces que componen el sistema y las relaciones que existen entre ellas
- Muestra aspectos estáticos
- Clase: conjunto de objetos que comparten:
  - Atributos
  - Operaciones
  - Relaciones
  - Semántica
- **Modelo de Dominio (Conceptual)**: ayudan a entender los conceptos del dominio del problema y el vocabulario del mismo. Se excluyen detalles referentes a la implementación o al lenguaje de programación.
- **Diagramas de clases de implementación**: muestran todos los métodos y atributos necesarios para implementar cada clase. Es un diagrama dependiente de la implementación y del lenguaje.

Modelo del Dominio (Conceptual)

- Permite describir las entidades que conforman el dominio, sus relaciones y atributos
- Se representan los conceptos del dominio
- Muestra aspectos estáticos

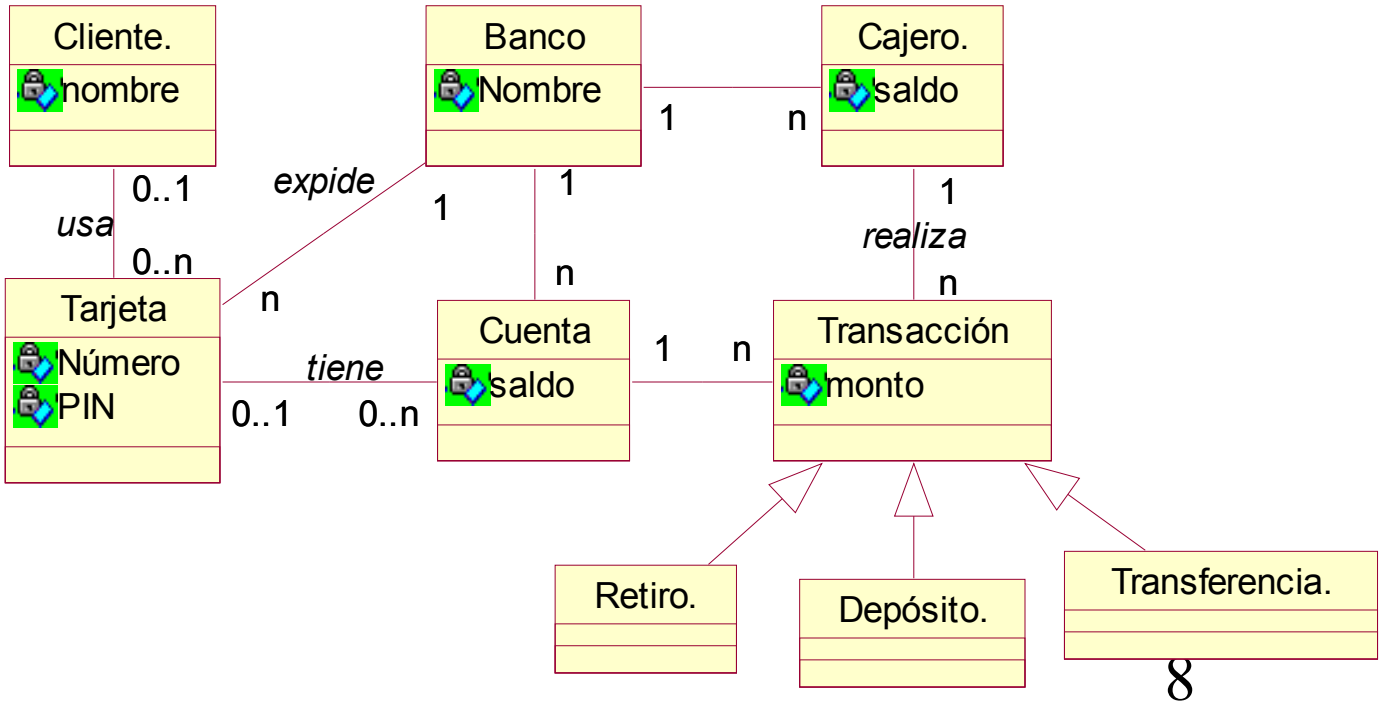


Diagrama de Actividad

- Se construye para modelar el flujo del control (workflow)
- Elementos:

Estado de Actividad (o de Acción)

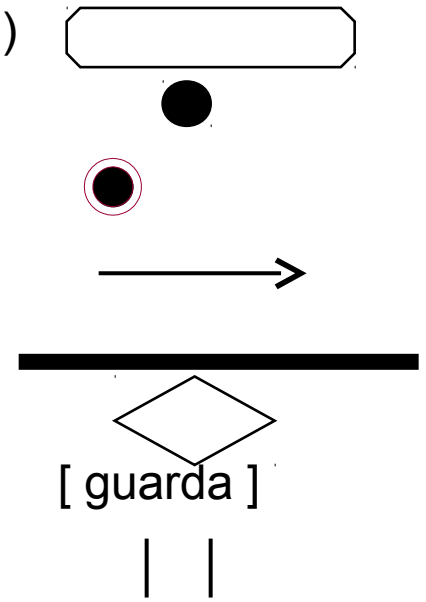
Estado Inicial

Transiciones

Actividades concurrentes

Bifurcaciones

Condiciones de la bifurcación



- Permite modelar el flujo del trabajo
  - En un sistema
  - En una organización

Diagrama de Actividad - Ejemplo

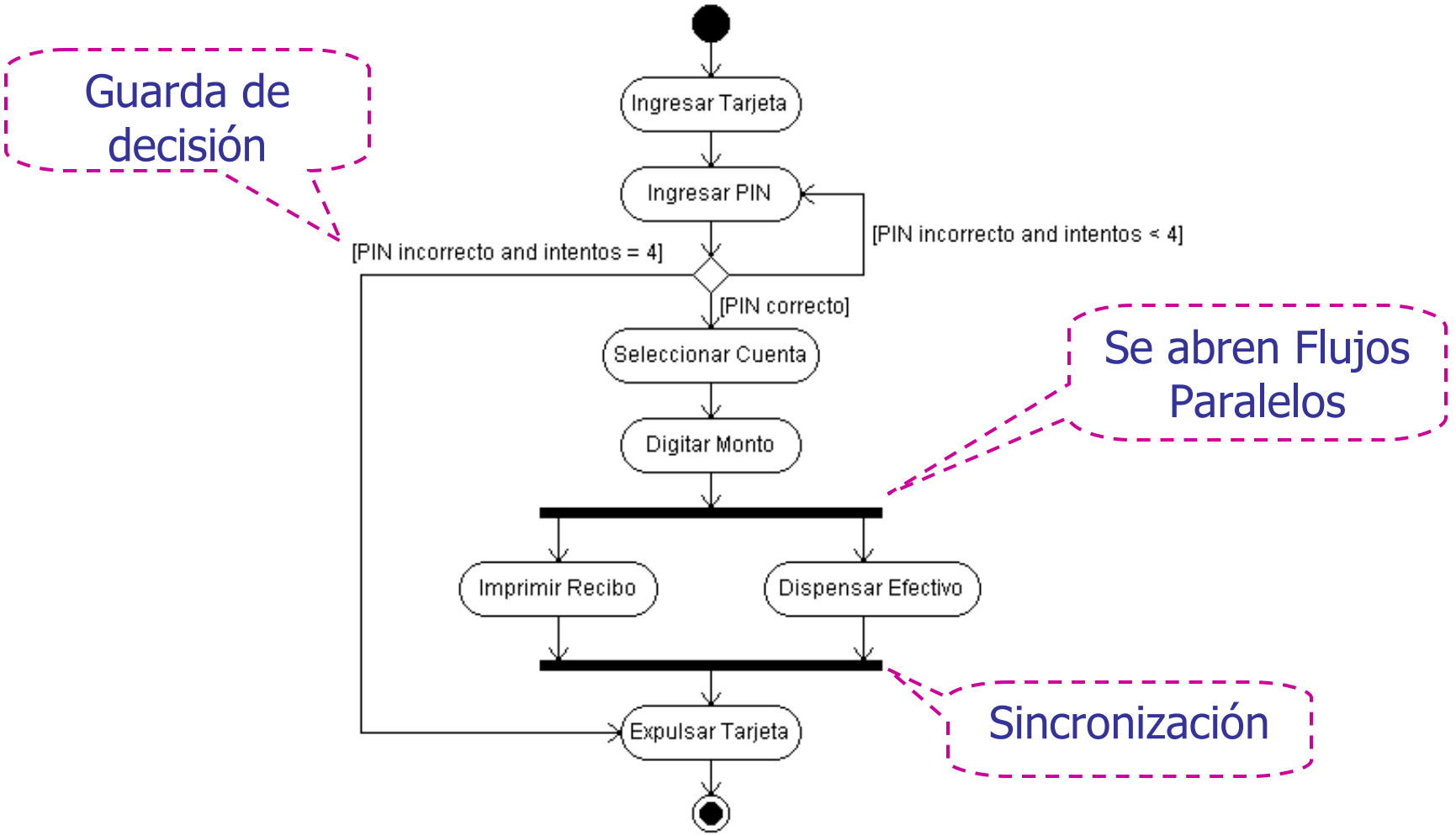


Diagrama de Máquinas de Estados

- Muestra el comportamiento de un objeto representando los estados en que se puede encontrar y los eventos que le hace pasar de uno a otro.
- Se utiliza para:
  - Modelar el estado interno de una entidad durante su ciclo de vida
  - Modelar el estado de un caso de uso
- Da una vista dinámica del sistema
- Permite:
  - Anidamiento: un estado con subestados
  - Estados paralelos: reduce el nro. de estados necesarios en el modelo
  - Condiciones de bifurcación

Diagrama de Estados

- Transición.
  - la etiqueta tiene tres partes optativas: *Evento {Guardia} / Acción*
- Los estados:
  - pueden tener actividades asociadas. Etiqueta con la sintaxis ***hace** / Actividad*.
  - estado inicial o de creación 
  - estado final – aquél que no tiene transiciones de salida 
- Las acciones:
  - se asocian con las transiciones
  - se consideran como procesos que suceden con rapidez y no se pueden interrumpir.
- Las actividades:
  - se asocian con los estados
  - pueden tardar más.
  - Una actividad puede ser interrumpida por algún evento.

Diagrama de Estados

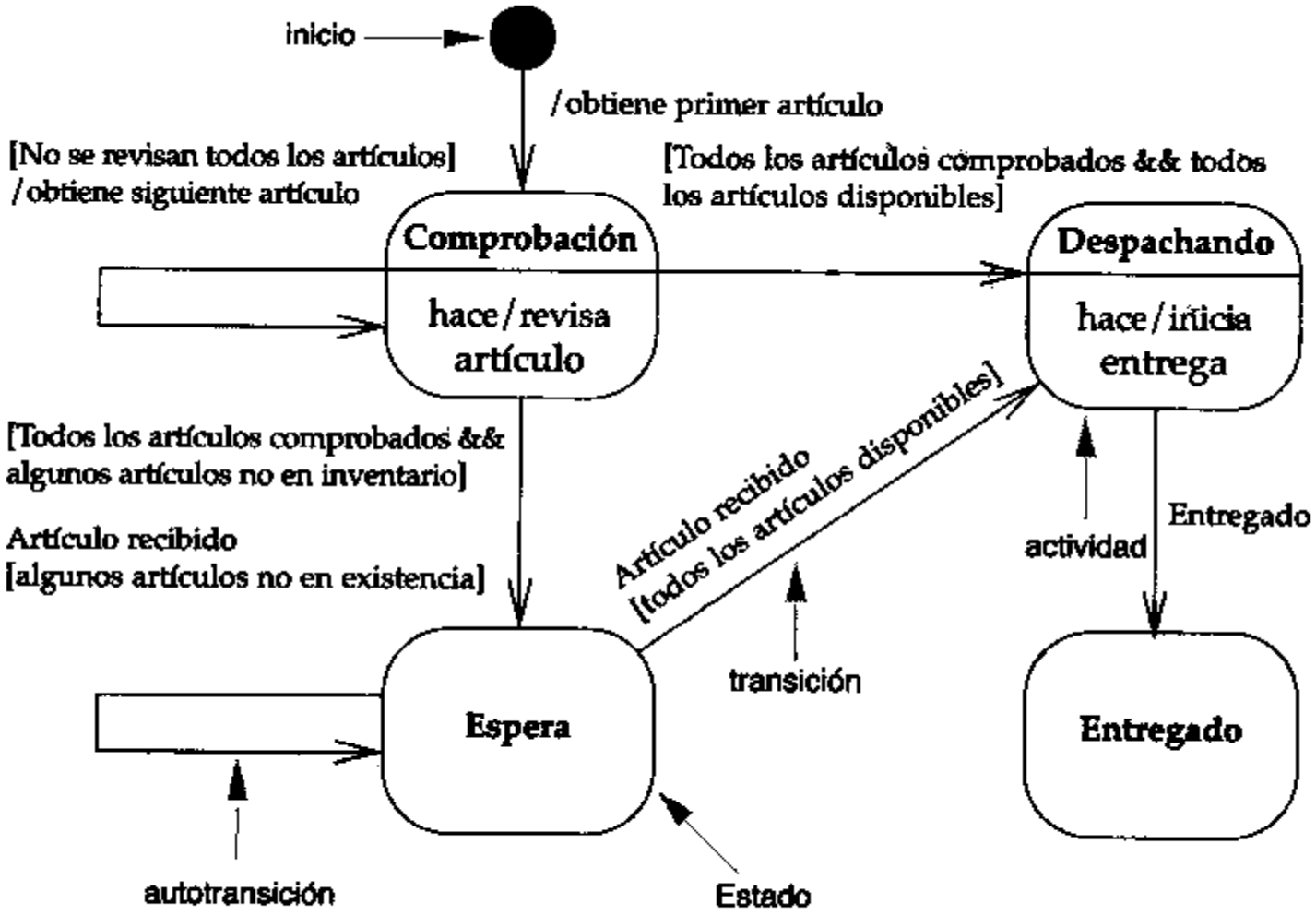


Figura 8-1: *Diagrama de estados*

Diagrama de Estados

- Una transición sin evento en su etiqueta se da tan pronto como se completa cualquier actividad asociada con el estado dado.
- Guardias:
  - Un guardia es una condición lógica que devuelve "verdadero" o "falso." Una transición de guardia ocurre sólo si el guardia es "verdadera".
  - Transición no guardada: la transición ocurrirá siempre que tenga lugar el evento.
  - Sólo se puede tomar una transición de un estado dado, por lo que los guardias deben **mutuamente excluyentes** para cualquier evento.
  - No es necesario que formen un conjunto completo.

Diagrama de Estados

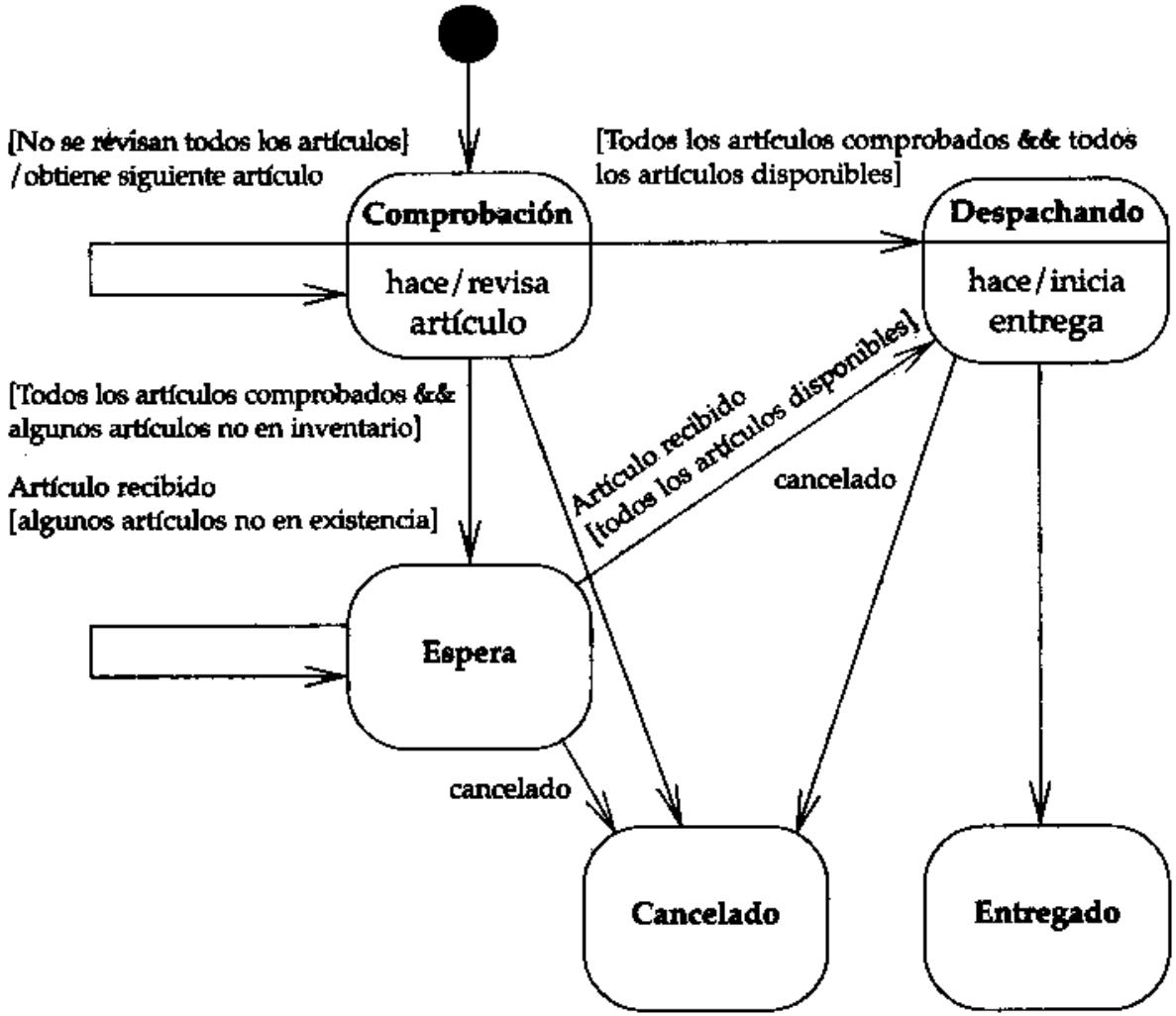


Figura 8-2: *Diagrama de estados sin superestados*

Diagrama de Estados - Superestados

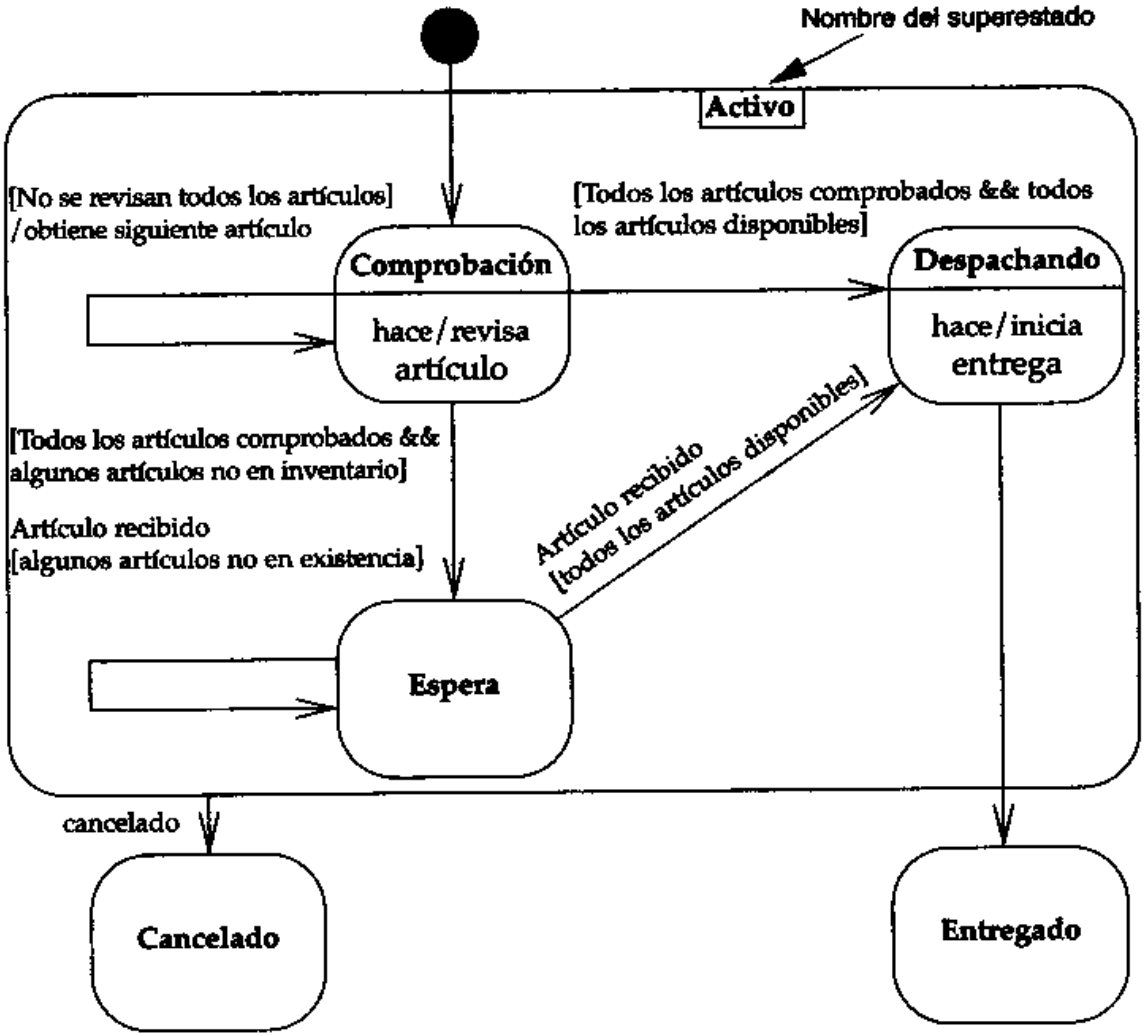


Figura 8-3: *Diagrama de estados con superestados*

- Los subestados heredan todas las transiciones sobre el superestado.
- A menos que haya un estado inicial.

Diagrama de Estados – Ejemplo 1

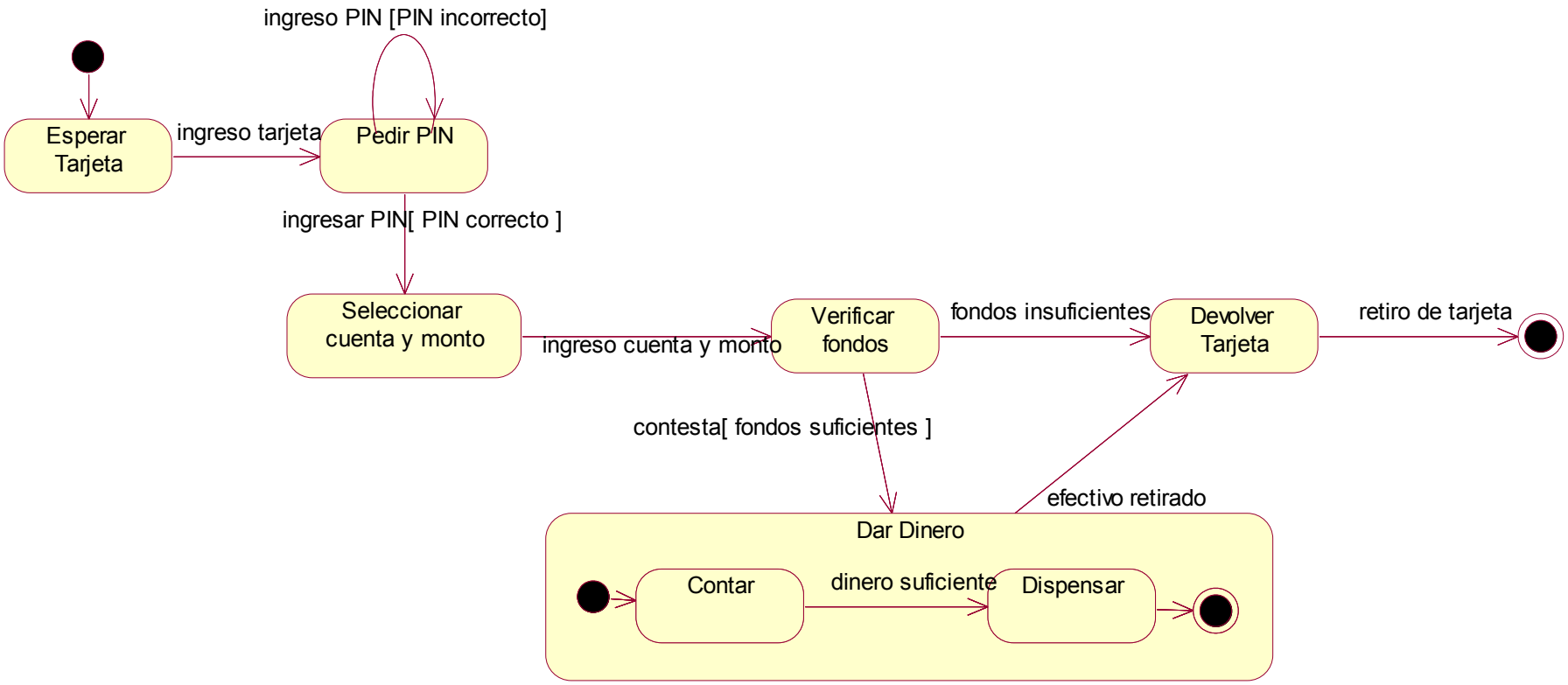


Diagrama de Estados

- Si un estado responde a un evento con una acción que no produzca una transición, se coloca *nombreEvento/nombreAcción* en el cuadro de estado.
- Existen también dos eventos especiales, entrada y salida.
  - Cualquier acción que esté vinculada al evento entrada se ejecuta siempre que se entre al estado. Sintaxis *entry/nombreAcción*
  - La acción asociada con el evento salida se ejecuta siempre que se sale del estado. Sintaxis *exit/nombreAcción*

  El poder indicar acciones al entrar o salir de un estado es útil porque evita documentar la misma acción para cada transición de entrada o salida del estado.
- En una autotransición se ejecuta:
  - la acción de salida
  - la acción de transición
  - la acción de entrada.
  - actividad asociada al estado

Diagrama de Estados

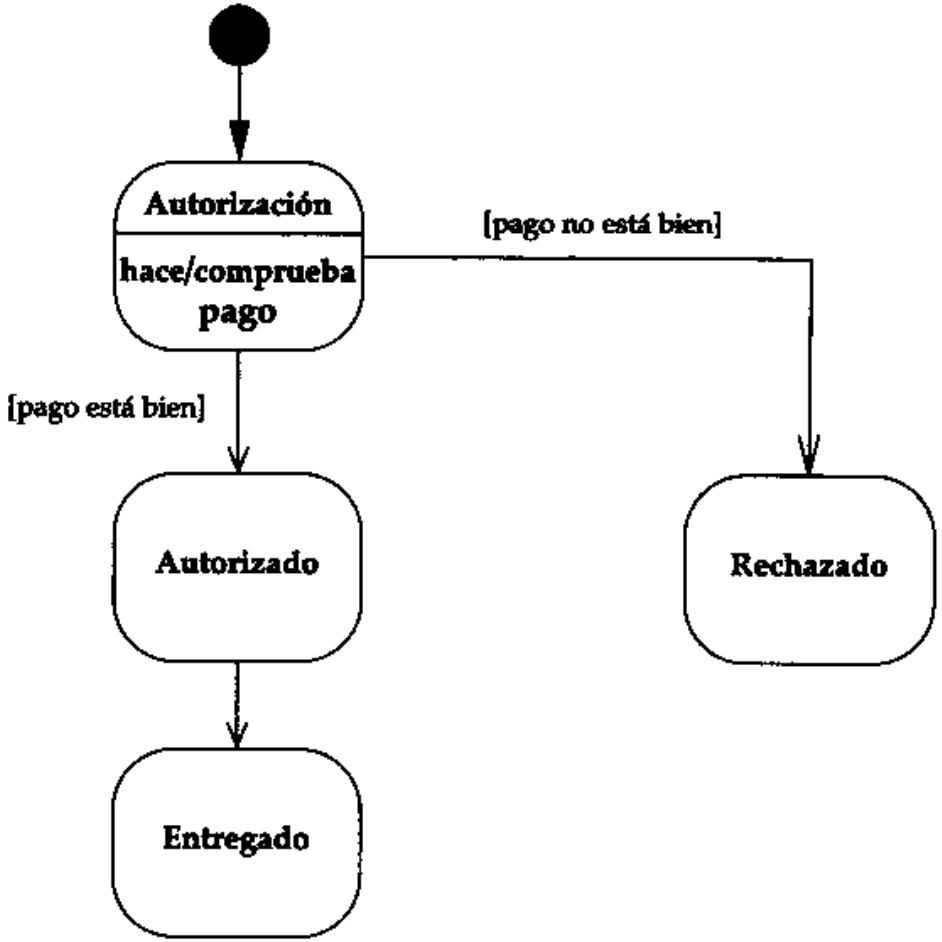


Figura 8-4: *Autorización de pagos*

Diagrama de Estados – Estados concurrentes

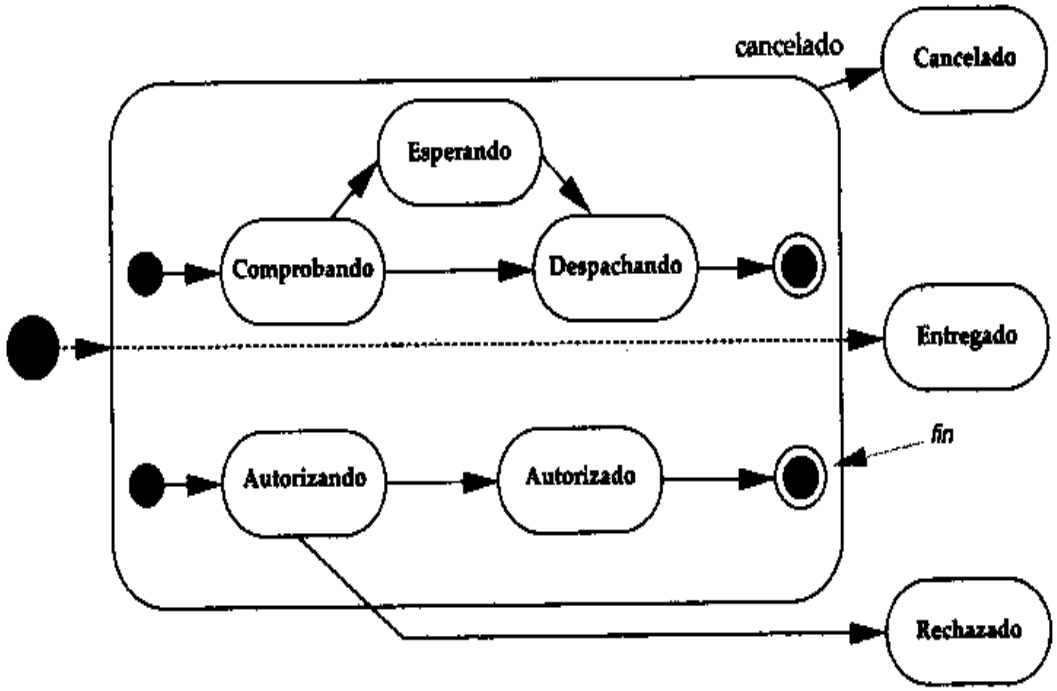


Figura 8-5: *Diagrama de estados concurrentes*

- Los diagramas de estados concurrentes son útiles cuando un objeto dado tiene conjuntos de comportamientos independientes.
- Recomendación:

  Si se tienen varios diagramas de estados concurrentes complicados para un solo objeto, considerar la división del objeto en varios.

Diagrama de Estados

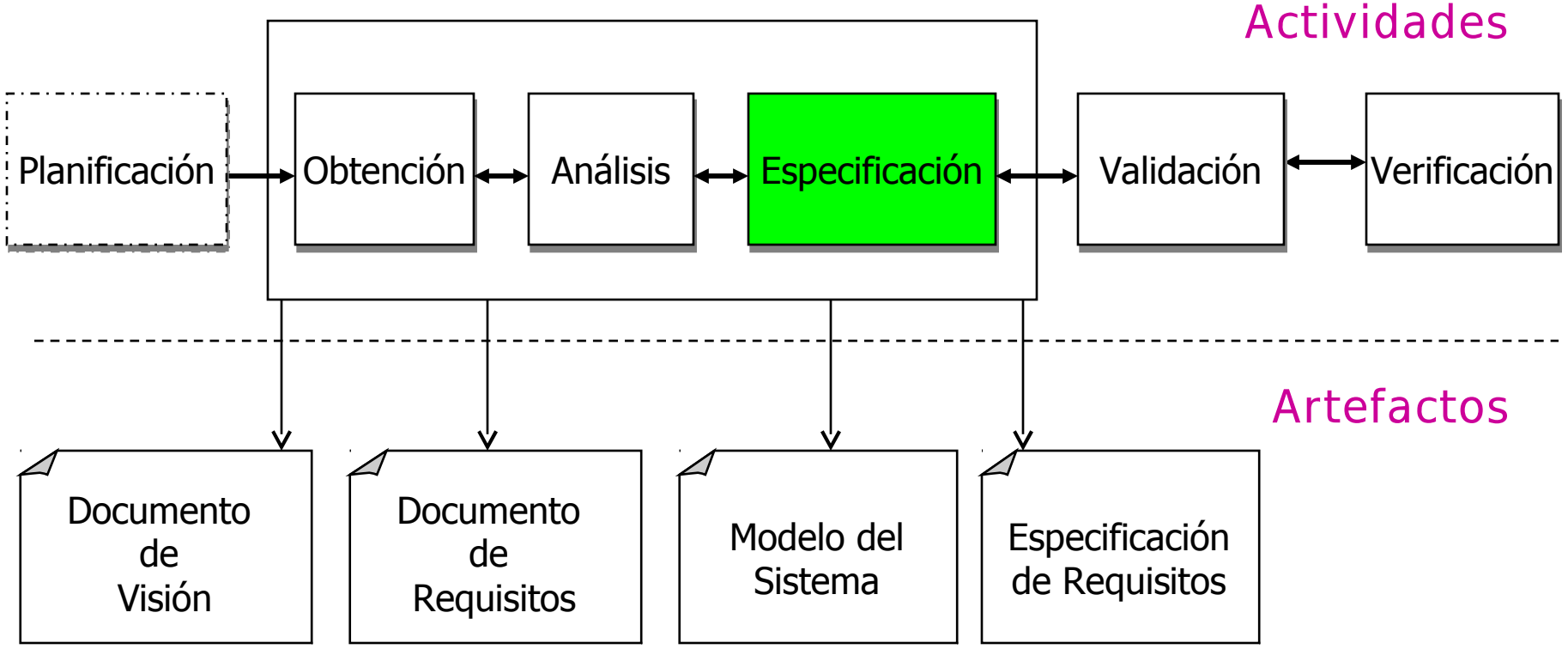
- Son buenos para describir el comportamiento de un objeto a través de varios casos de uso.
- No son tan buenos para describir un comportamiento que involucra cierto número de objetos que colaboran entre ellos.

Elección de una Técnica para Modelar Requisitos

- No existe un único enfoque aplicable a todos los sistemas, depende de cada proyecto
- Puede ser necesario combinar varios enfoques

Especificación de Requisitos

Proceso de Requisitos



Lenguajes de Notación

- Lenguaje Natural
  - Comprensible para el Cliente/Usuario
  - Ambiguo (glosario)
  - Poca legibilidad (plantilla, formateo del texto)
  - Difícil de tratar (Verificar correctitud, consistencia, completitud)
- Notaciones Especiales (más formales)
  - Poca o ninguna ambigüedad
  - Facilita tratamiento
  - Necesidad de entrenamiento en la notación
  - Dificultades de comprensión por Cliente/Usuario

Notaciones Especiales

- Gráficas vs. Basadas en texto
- Estáticas vs. Dinámicas
- Descripciones Estáticas
  - Se especifican entidades y sus atributos, los requisitos se pueden ver como las relaciones entre las entidades.
  - No describe como cambian las relaciones con el tiempo.
- Descripciones Dinámicas
  - Especifican estados y las transiciones entre estados en el tiempo.

Documentación de requisitos

- Qué documentar:
  - [illegible]
  - [illegible]
  - [illegible]
  - [illegible]
  - [illegible]
- Recomendaciones:
  - [illegible]
  - [illegible]
  - [illegible]
  - [illegible]
  - [illegible]

Documentos de Requisitos

- **Definición de Requisitos**: lista completa de lo que el cliente espera que el sistema haga, escrita de forma que el cliente la pueda entender. Ejemplo:
  1. "Se debe proveer un medio para acceder a archivos externos creados por otras herramientas."
- **Especificación de Requisitos (SRS)**: reformula la definición en términos técnicos para que los diseñadores puedan comenzar el diseño. Ejemplo:

  "1.1 Se proveerá al usuario los recursos para definir el tipo de archivo externo."

  "1.2 Cada tipo de archivo tendrá una herramienta asociada y un ícono que lo identifica."

  "1.3 Cuando el usuario seleccione el ícono que representa un archivo externo, el efecto es aplicar la herramienta asociada con ese tipo de archivo al archivo seleccionado."

Documentos de Requisitos (2)

- Usar un mismo documento: Entendimiento común entre Cliente, usuario, analistas, desarrolladores.
- Usar dos documentos:

  Se debe aplicar **Gestión de la Configuración**:
  - Es necesaria para asegurar la correspondencia entre ambos (si existen por separado).
  - Permite seguir la pista y correspondencia entre:
    - Definición de Requisitos
    - Especificación de Requisitos
    - Módulos de Diseño
    - Código que implementa los módulos
    - Pruebas para verificar la funcionalidad
    - Documentos que describen el sistema

Documento Definición de Requisitos

- Registrar los requisitos en los términos del cliente:
  - 1. Delinear el propósito general del sistema: Incluir referencias a otros sistemas, glosario y abreviaciones.
  - 2. Describir el contexto y objetivos del desarrollo del sistema.
  - 3. Delinear visión global del sistema: Incluir restricciones generales.
  - 4. Definir en detalle las características del sistema propuesto, definir la frontera del sistema e interfaces.
  - 5. Discutir el ambiente en el que el sistema va a operar (hardware, comunicaciones, personal).

Características de una Buena Especificación SRS (IEEE 830)

- Correcta / Válida: Todos los req. son requeridos en el sistema.
  - No existe herramienta que asegure esto.
  - Validado por el cliente (que efectivamente refleje sus necesidades).
  - Revisar que sea consistente contra otros documentos existentes (pe. especificación de reqs. del sistema).
- No Ambigua: Todo req tiene una única interpretación.
  - Incluir glosario.
  - No ambigua para quienes lo crearon y para quienes lo usan.
- Completa: Incluye:
  - Todos los requisitos asociados con funcionalidad, desempeño, restricciones de diseño, atributos o interfaces externas.
  - Definición de respuestas del sw a todo posible datos de entrada (válidos o inválidos) en toda clase de situaciones realizables.
  - No hay referencias sin definir en la especificación.
  - La frase “a determinar” indica SRS no completa. Ocasionalmente necesaria; describir:
    - condiciones que causan que no se sepa aún.
    - qué se debe hacer para determinar lo que falta, quién y cuándo.

Características de una Buena Especificación SRS (IEEE 830)

- Consistente internamente: Los requisitos no son contradictorios entre sí. Probables conflictos:
  - entre características de entidades. Pe. color de las luces, formatos distintos
  - conflicto lógico o temporal entre dos acciones. Pe. multiplicar o sumar; en forma simultánea o consecutiva.
  - diferentes términos para describir el mismo objeto.
- Ordenados por grado de importancia y/o estabilidad – identificador.
  - Importancia: esencial / deseado
  - Estabilidad: cantidad de cambios esperados
  - Necesidad: esencial / condicional / opcional
    - Esencial (condiciona aceptación del sw)
    - Condicional (valor agregado)
    - Opcional (puede o no valer la pena; se aceptan propuestas alternativas).

Características de una Buena Especificación SRS (IEEE 830)

- Verificable: Un requerimiento es verificable si existe un proceso finito de costo accesible para determinar que el sistema lo cumple.
  - Usar términos concretos y cantidades mesurables.
  - Preparar pruebas para demostrar que se cumplen. Si no se puede, eliminar o revisar el requisto.
- Modificables: Su estructura y estilo son tales que cualquier cambio en los requisitos puede ser hecho fácilmente en forma completa y consistente.
  - Organización coherente y fácil de usar (tablas, índices, refs. cruzadas
  - No redundante.
    - Ventajas de redundancia: lo hace más legible.
    - Desventajas: difícil de mantener
    - Si la uso: referencias cruzadas
  - Expresar cada req. separadamente.

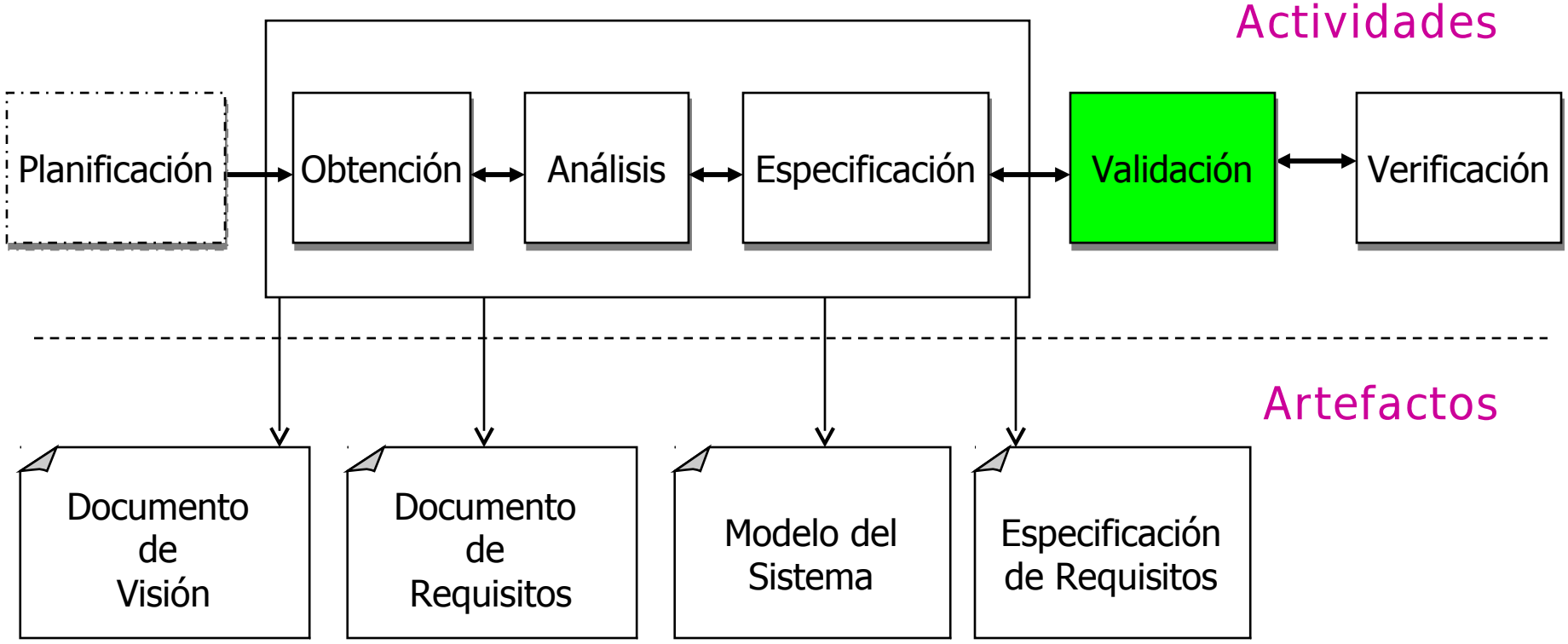
Características de una Buena Especificación SRS (IEEE 830)

- Trazables: El origen de cada requerimiento es claro, y es posible seguirle la pista en futuros desarrollos o mejora de la documentación.
  - Trazabilidad hacia atrás: en versiones previas
  - Trazabilidad hacia adelante: documentos posteriores:
    - requiere IDENTIFICADOR ÚNICO.

- Realistas / Factibles
  - Ej.: tiempo de respuesta local=remoto
  - Ej.: El cliente quiere adelantarse a la tecnología
- Entendibles: Tanto por los usuarios como por los desarrolladores

Validación de Requisitos

Proceso de Requisitos



Validación de Requisitos

- Proceso por el cual se determina si los requisitos relevados son consistentes con las necesidades del cliente.
- Objetivo:
  - Asegurar que se esté construyendo el sistema correcto.
- Requisitos sirven como:
  - contrato con el cliente
  - guías para los diseñadores.
- Proceso:
  - Planificar quién (qué stakeholder) va a validar qué (artefacto) cómo (técnica).
  - Ejecutar
  - Registrar – Reporte de validación / Firma

Validación de Requisitos

- Se chequea en el documento de requisitos:
  - Validez: que el usuario valide qué es lo que quiere.
  - Consistencia: que no haya contradicciones
  - Completitud: que no falte nada. Chequear por:
    - [redacted]
    - [redacted]
  - Necesidad
  - Ambigüedades
  - Realismo o Factibilidad: que se puedan implementar con la tecnología, presupuesto y calendario existentes.
  - Verificabilidad: que se pueda diseñar conjunto de pruebas para demostrar que el sistema cumple esos requisitos. Cuidado con adjetivos y adverbios.
  - Comprensibilidad: que los usuarios finales lo entiendan
  - Adaptabilidad: que el requisito se pueda cambiar sin afectar a otros.
  - Trazabilidad: que esté establecido el origen.

Validación de Requisitos NO Funcionales

- Son difíciles de validar.
- Se deben expresar de manera cuantitativa utilizando métricas que se puedan probar de forma objetiva (esto es IDEAL).

Propiedad	Medida
Rapidez	Transacciones por seg
Tamaño	KB
Fiabilidad	Tiempo promedio entre fallas
Portabilidad	Número de sistemas, especificar
Facilidad de uso	Tiempo de capacitación

- Para los usuarios es difícil especificarlos en forma cuantitativa.

Técnicas de Validación

- Manuales
  - [illegible]
  - [illegible]
  - [illegible]
  - [illegible]
  - [illegible]
    - Revisiones - Stakeholders revisan por separado y se reúnen para discutir problemas.
    - Inspecciones formales – roles y reglas.
  - [illegible]
  - [illegible]
  - [illegible]
- Automatizadas
  - [illegible]
  - [illegible]
  - [illegible]
  - [illegible]

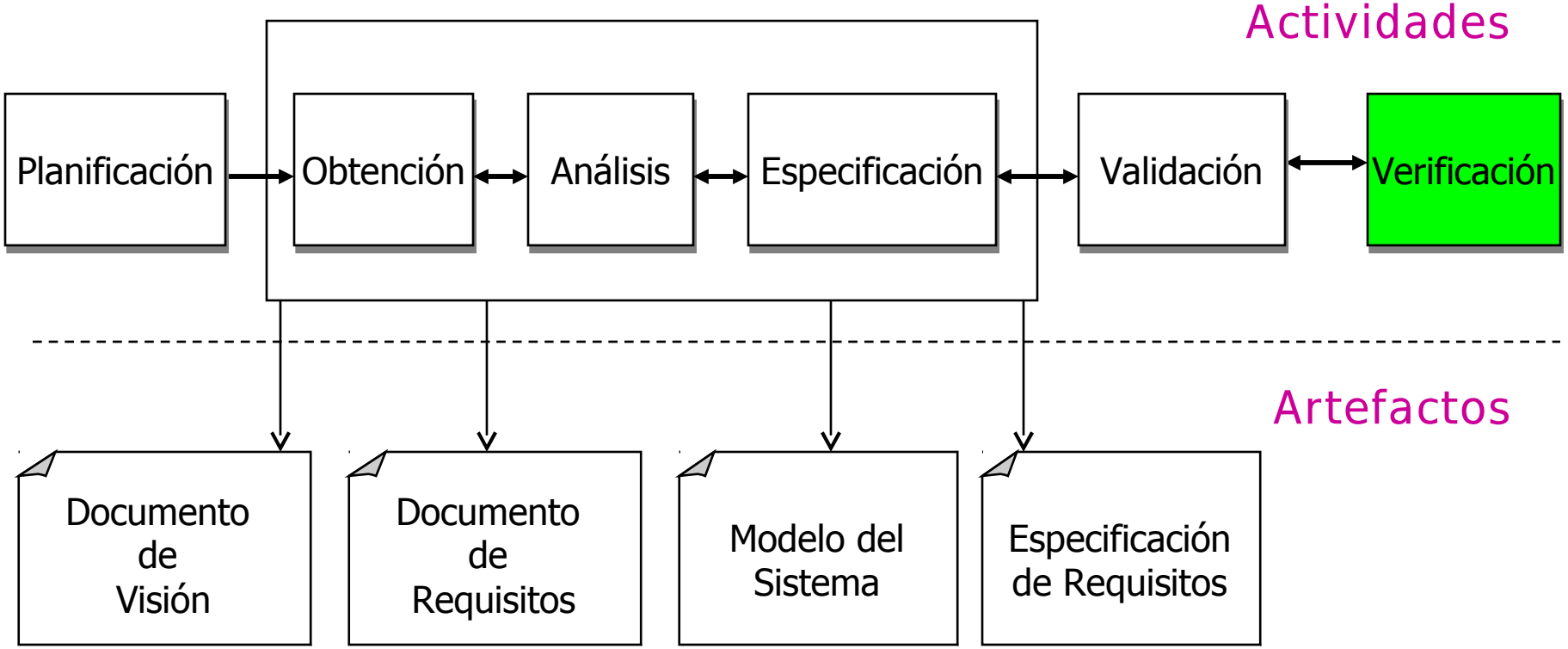
Revisión de Requisitos

- Proceso manual. Se revisa el documento de requisitos buscando anomalías y omisiones:
  - [redacted]
  - [redacted]
- Participan representantes:
  - [redacted]
  - [redacted]
- Incluye:
  - [redacted]
  - [redacted]
  - [redacted]
  - [redacted]
  - [redacted]
  - [redacted]
    - pruebas del sistema.
    - cambios en los requisitos en el proyecto, su verificación y validación.

¿Cómo asegurar que la reunión es efectiva? Moderador, secretario y responsables por acciones

Verificación de Requisitos

Proceso de Requisitos



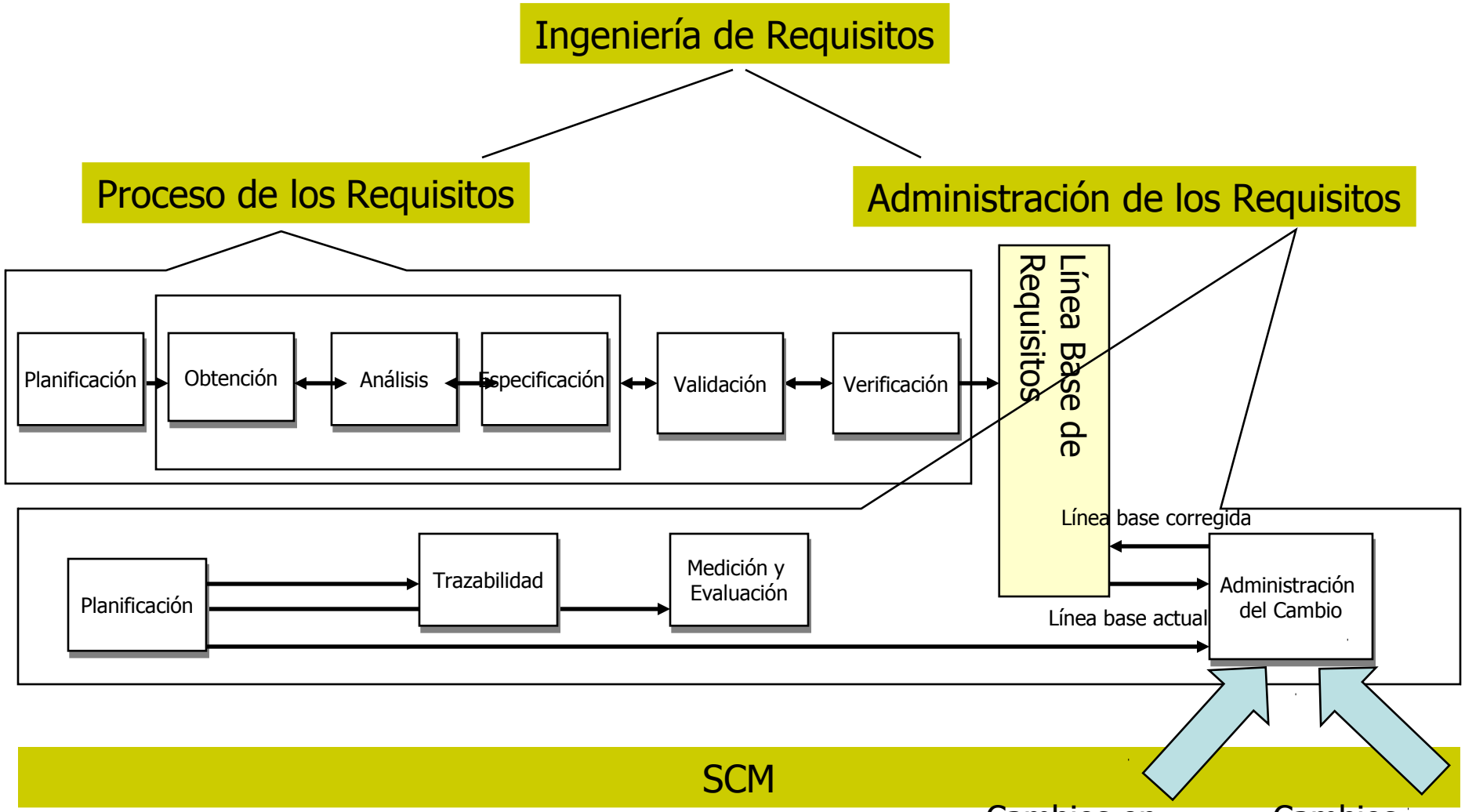
Verificación de requisitos

- Objetivo:
  - Asegurar que se esté construyendo el sistema correctamente.
  - Se verifica que un artefacto (salida) sea conforme a otro (entrada).
- Usualmente es solo chequeo de trazabilidad de la especificación al documento de reqs.
- Para sistemas críticos: demostrar que la especifición realiza los requisitos:
  - usamos SRS + asunciones sobre el comportamiento del ambiente (¡documentarlas!)

Verificación de requisitos

- Chequeos:
  - chequeo de referencias cruzadas
  - chequeos de consistencia
  - chequeos de completitud
  - chequeos por estados o transiciones inalcanzables.
- Técnicas:
  - revisiones formales (en grupo)
  - revisiones por pares
  - listas de comprobación
  - modelos
  - Pruebas Matemáticas: Si se usó un lenguaje formal, pe. Z.

Ingeniería de Requisitos



Administración de los Requisitos

- Los requisitos cambian, debido a:
  - Muchos usuarios
  - Quienes pagan por el sistema y los usuarios no son las mismas personas
  - Cambios en el negocio
  - Cambios en la tecnología
- Proceso de comprender y controlar los cambios en los requisitos del sistema.
- Se hace en paralelo con el Proceso de Requisitos.
- Tres etapas:
  - Planificación: Se realiza al comenzar el análisis de requisitos
  - Administración del cambio: Comienza una vez que se tiene una primera versión del documento de requisitos
  - Trazabilidad: Se mantiene a lo largo del proceso de requisitos

Planificación

- Muchas actividades son tomadas de las técnicas de SCM.
- Se debe decidir sobre:
  - **Identificación de Requisitos**: Cada requisito debe identificarse en forma única, para poder ser referenciado por otros.
    - [illegible]
      - [illegible]
      - [illegible]
  - **Procedimiento de Administración del Cambio**: Actividades que evalúan el impacto y costo del cambio.
  - **Políticas de Trazabilidad**: Definen qué relaciones entre reqs. y con el diseño se deben registrar y cómo se van a mantener.
  - **Herramientas CASE**: De soporte para:
    - [illegible]
    - [illegible]
    - [illegible]

Trazabilidad

- Información de rastreo que se debe mantener
  - **La fuente**: Quién propuso el requerimiento y porqué.
  - **Requisitos dependientes**: Vincula los requisitos dependientes entre sí; se usa para el análisis del cambio.
  - Trazabilidad **entre artefactos** distintos, qué versión se corresponde con cuál. Pe.:
    - Rastreo reqs - CU
    - **Rastreo al diseño:** Vincula el req con los módulos de diseño que lo implementan

- Uso de matrices de trazabilidad

Administración del Cambio

- El cambio va a ocurrir.
- Objetivos del control de cambios:
  - Manejar el cambio y asegurar que el proyecto incorpora los cambios correctos por las razones correctas.
  - Anticipar y acomodar los cambios para producir la mínima disrupción y costo.
- Si los reqs cambian mucho dp de LB →
  - relevamiento incompleto/inefectivo
  - o acuerdo prematuro

Administración del Cambio

- Cuando se propone un cambio, debe evaluarse el impacto.
- Etapas:
  1. Especificación del cambio.
  2. Evaluar impacto - Análisis del cambio y costo:
     - Se usa la información del rastreo
     - Se calcula el costo en términos de modificaciones a:
       - Docs de requisitos
       - Diseño e implementación
  1. Discutir costo con cliente.
  2. Implementar el cambio: se modifican los artefactos necesarios.

- Siguiendo estos pasos se logra
  - Todos los cambios se tratan en forma consistente.
  - Los cambios a los docs de requisitos se hacen en forma controlada.

Procedimiento de control de cambios

- Establecer procedimiento de control de cambios:
  - quién - Comité de Control de Cambios (CCC)
  - documentar:
    - integración del CCC
    - alcance de autoridad
    - procedimientos operativos (pe. evaluar impacto)
    - proceso de toma de decisiones

Gestión de la Configuración de los Requisitos

- Tiene que haber un responsable
- Control de versiones: Definir:
  - Ítems de configuración
  - Procedimientos
- Línea Base. Definición:
  - Conjunto de especificaciones y /o productos que han sido revisados formalmente y acordados, que sirven de base para desarrollo futuro, y que solo pueden ser cambiados a través de procedimientos formales de control de cambios.

Línea Base de Requisitos

- LB de reqs, arranca cuando se decide que son suficientemente buenos como para arrancar diseño y construcción.
- Sobre LB planifico cronograma y costo.
- Asociada a la liberación de un producto. Debo poder recomponer la liberación.
- Definir:
  - qué artefactos van en Línea Base
  - cuándo entran

Medir y Evaluar Requisitos

- Medir características de los requisitos para obtener detalles
  - Proceso de los Requisitos
  - Calidad de los Requisitos
- Las mediciones van a estar relacionadas con:
  - Producto (de los requisitos)
    - tamaño, calidad, atributos técnicos, ...
  - Proceso
    - actividades,...
  - Recursos
    - personas, equipos, tiempo, dinero,...

Medir y Evaluar Requisitos

- Medir
  - # Requisitos
    - Entrada para estimación del producto
  - # Cambios introducidos
    - Requisitos Agregados, Modificados, Desechados en el tiempo
    - Estabilidad
  - # Requisitos por tipo de requisitos
    - Permite luego ver en qué parte se encuentra el cambio
  - # Requisitos validados
- Tamaño del producto y del proyecto (ej.:PF, LoC)
    - planificar

Metodologías de Desarrollo

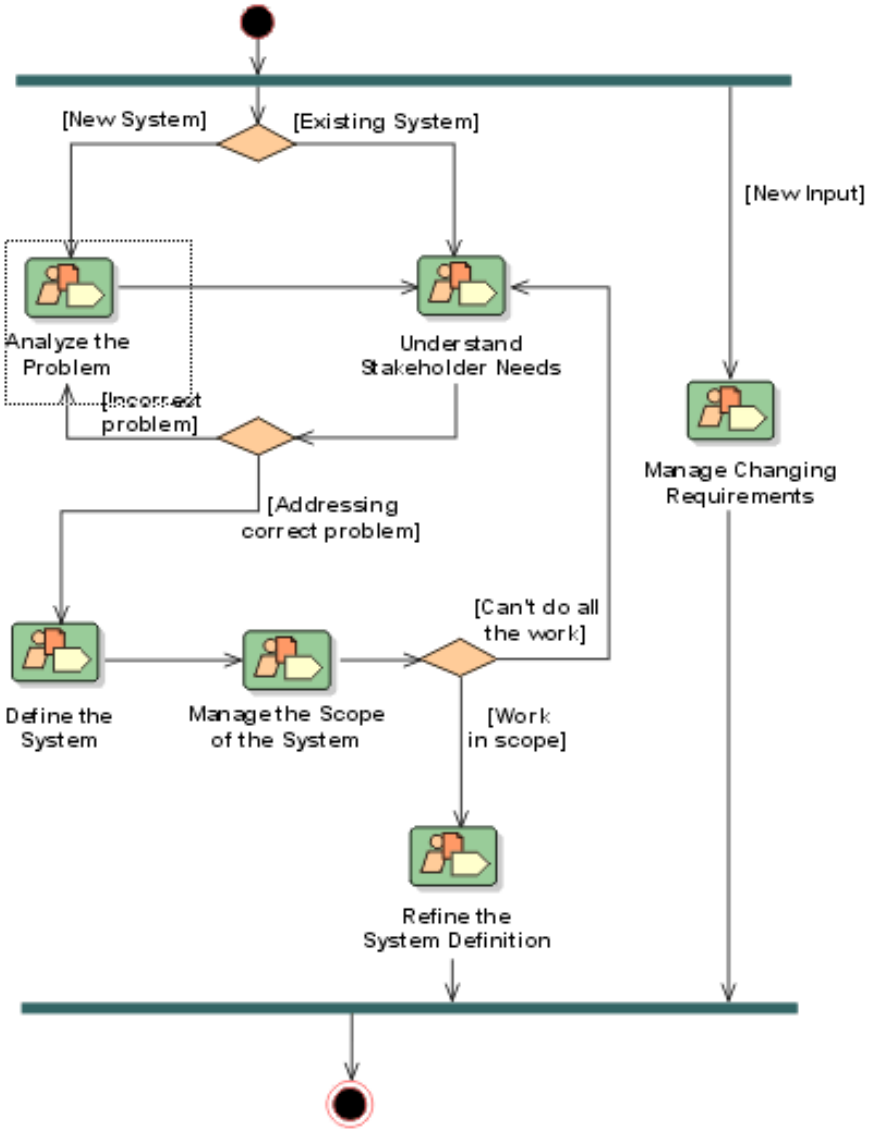
Metodologías de Desarrollo

- Los métodos ágiles fueron desarrollados en respuesta a la necesidad de tener una alternativa a los procesos de desarrollo de software pesados, “guiados por documentos” (AgileAlliance 2002).
- Cada metodología trata distinto los requisitos.
- Ejemplo de metodología ágil: eXtreme Programming (XP)
- Ejemplo de metodología pesada: Rational Unified Process

eXtreme Programming

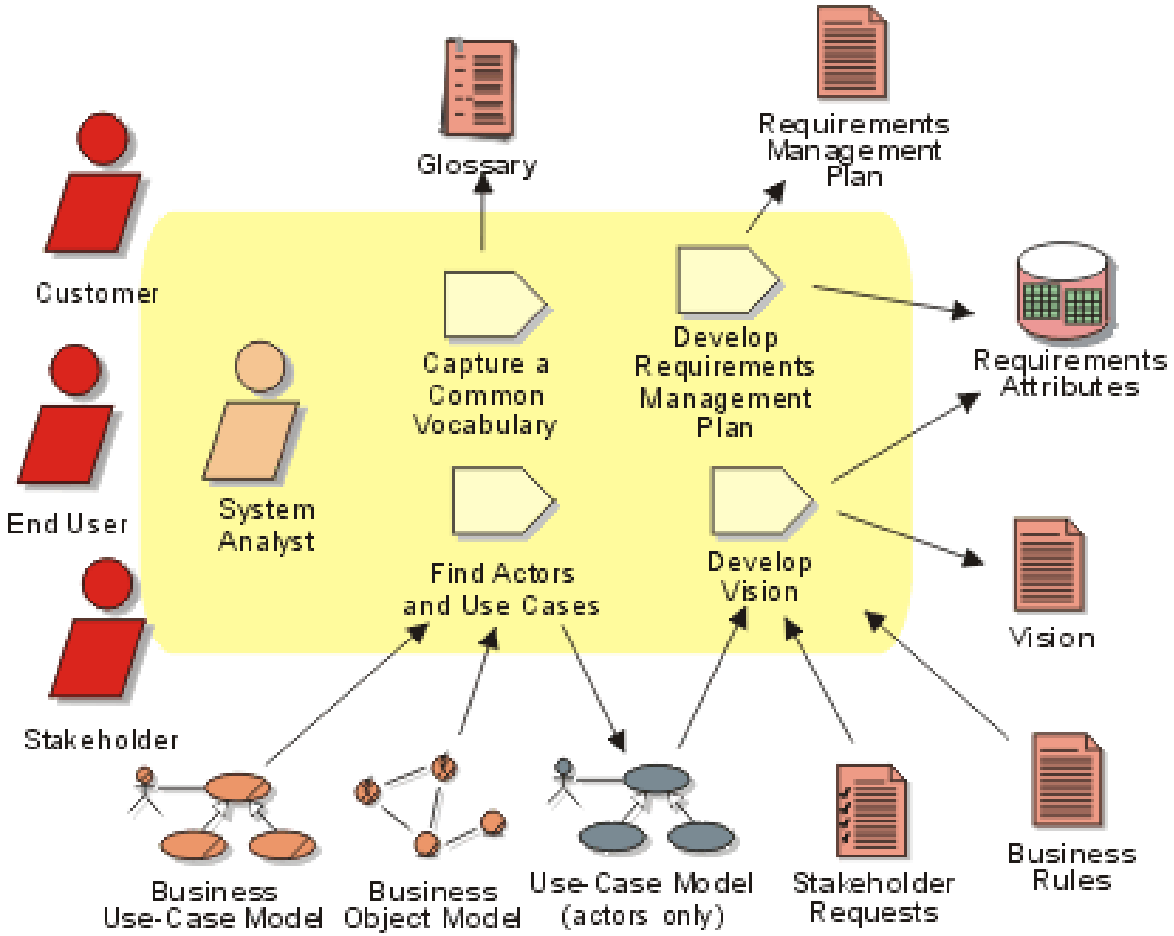
- Cliente en el lugar
  - Un cliente real debe sentarse en el lugar, disponible para escribir historias, contestar preguntas, resolver disputas y prioridades de pequeña escala.
- Historias de usuario
  - Su propósito es análogo al de los casos de uso.
  - Son escritas por el cliente, son las cosas que el sistema debe hacer.
  - No son casos de uso, pero describen escenarios.
  - Su formato son tres sentencias de texto escritas por el cliente, en su terminología sin sintaxis técnica.
  - Cuando llega el momento de implementarla, los dasarrolladores van con el cliente y reciben una descripción detallada de los requisitos, cara a cara.
  - Se usan para planificar el proyecto: se estiman y el cliente las prioriza definiendo el alcance.

Rational Unified Process – Disciplina de Requisitos



RUP – Detalle de Actividades

Workflow Detail: Analyze the Problem



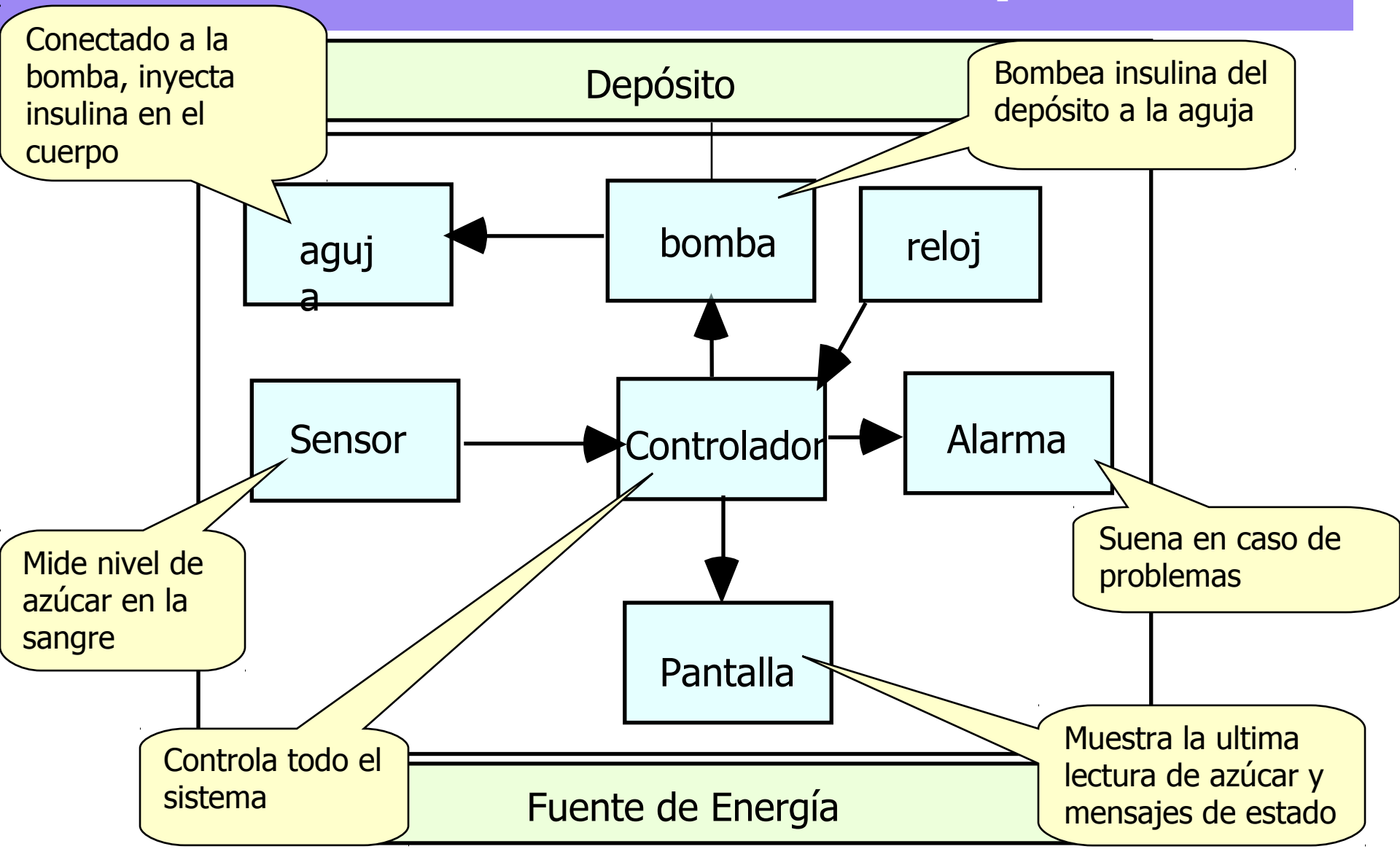
Especificación Formal

- Ventajas:
  - Permite detectar omisiones e inconsistencias en los requisitos.
  - Detección temprana de defectos.
  - Necesario para demostrar que un programa es correcto.
- Desventajas:
  - Exige entrenamiento.
  - En general no es comprensible para el cliente.
  - Funcionan bien en escalas reducidas, pero se complican a medida que crece la escala del producto.
- Su aplicación suele estar restringida a sistemas críticos.

Sistema crítico, un ejemplo

- Bomba personal de insulina que intenta emular el comportamiento del páncreas (alternativa frente a inyecciones de insulina).
- Un sensor mide el nivel de azúcar en la sangre.
- Requisitos relativos a la criticidad:
  - Disponibilidad – el sistema debe funcionar cuando el paciente necesite insulina.
  - Confiabilidad – debe proveer insulina en momento y cantidad adecuada.
  - Seguridad (safety) – una dosis excesiva podría poner en riesgo la vida del paciente.

Bomba de Insulina - Componentes



Concepto de Operación

- A partir de la lectura del sensor, el sistema evalúa el nivel de glucosa en sangre del paciente.
- El sistema compara lecturas consecutivas para detectar una posible tendencia al crecimiento del nivel. En este caso inyecta insulina para actuar en contra de esa tendencia.
- La situación ideal es que el nivel de azúcar se encuentre sistemáticamente en la banda de seguridad.
- Niveles de azúcar en sangre:
  - Inseguro – menos de 3 unidades (posible coma)
  - Seguro- entre 3 y 7 unidades
  - No deseable (más de 7 unidades)

Nota: los valores mencionados sólo son a título ilustrativo.

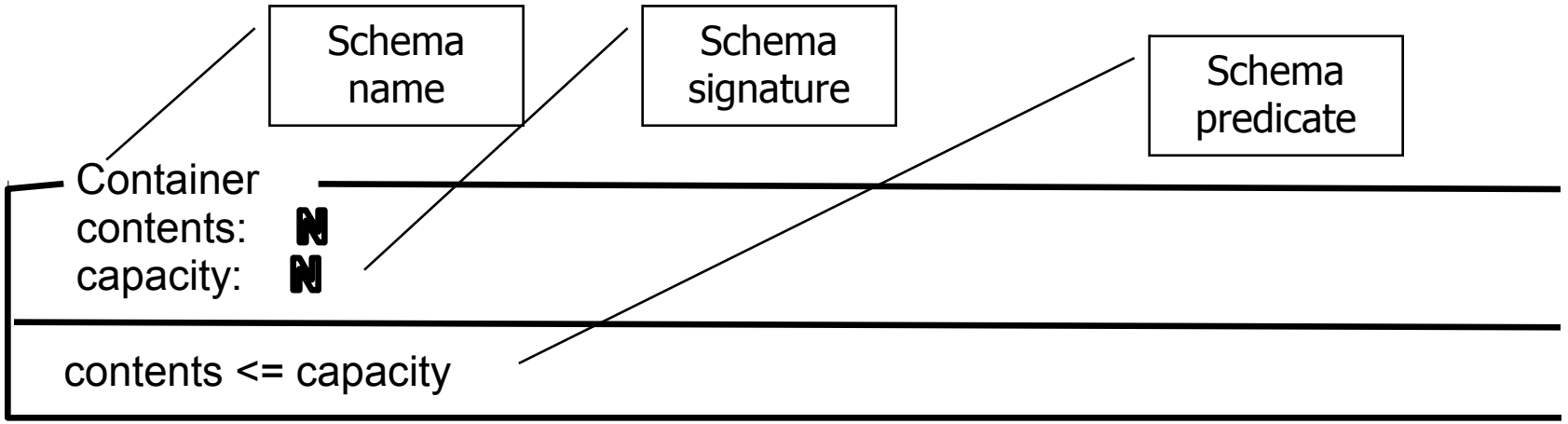
Inyección de Insulina

- Según nivel de azúcar, tendencia e inyecciones anteriores
- Escenarios relativos al nivel de azúcar en sangre:
  - en la banda insegura
    - No inyectar
    - Alarma para el paciente
  - en las otras dos bandas
    - cayendo
      - [illegible]
      - [illegible]
    - estable
      - [illegible]
      - [illegible]
    - creciendo
      - [illegible]
      - [illegible]

Lenguaje de Especificación Z

- Se han desarrollado diversos lenguajes y herramientas para la especificación (y verificación) de software.
- Z (Hayes 87, Spivey 92) está basado en la teoría de conjuntos tipados
  - modela un sistema en base a conjuntos y sus relaciones
  - introduce elementos que facilitan la especificación de pre y post condiciones asociadas a estados
  - los modelos se construyen a partir de “esquemas”
- Esquema: Introducen variables de estado y definen restricciones y operaciones sobre los estados.
- Una especificación es representada como un conjunto de esquemas.
- Los esquemas pueden ser combinados y usados en otros esquemas.

Lenguaje de Especificación Z



- Schema signature:
  - Declara nombres y tipos de las entidades.
- Schema predicate:
  - Define relaciones entre las entidades de la signature mediante expresiones lógicas que deben ser verdaderas (invariantes).

Lenguaje de Especificación Z

- Nombres seguidos por ? son entradas y por ! , salidas.
- Nombre seguido por ‘ significa el valor después de la operación.
- ∀ Ξ precediendo a un nombre significa que los valores no son cambiados por la operación.
- ∀ Δ precediendo a un nombre significa que los valores son cambiados por la operación.
- Incluir el esquema A en el esquema B significa que B hereda los nombres y predicados de A.

Z para la bomba de insulina

- Conjunto de variables de estado:
  - reading? : Lectura del sensor de glucosa en la sangre.
  - dose, cumulative_dose: dosis a suministrar y dosis acumulada en un período.
  - r0, r1, r2: 3 últimas lecturas, se usa para calcular la razón del cambio de glucosa en la sangre.
  - Capacity: capacidad del depósito.
  - alarm!: alarma.
  - pump!: señal de control enviada a la bomba.
  - display1!, display2! : msg de estado y dosis a administrar.
- Para la bomba de insulina:
  - dosis<=contenido del depósito.
  - dosis<=5 unidades y suma de dosis en período <=50.
  - display1! muestra el estado del depósito.

Esquema para la bomba de insulina

Insulin_pump

reading? : ℕ
dose, cumulative_dose: ℕ
r0, r1, r2: ℕ // para registrar las 3 últimas lecturas tomadas
capacity: ℕ
alarm!: {off, on}
pump!: ℕ
display1!, display2!: STRING

---

dose<=capacity ∧ dose<=5 ∧ cumulative_dose<= 50
capacity>=40 ⇒ display1! = " "
capacity<=39 ∧ capacity>=10 ⇒ display1! = "Insulina baja"
capacity<=9 ⇒ alarm! = on ∧ display1! = "Insulina muy baja"
r2 = reading?

Cálculo de la dosis

DOSAGE
ΔInsulin_Pump

Delta indica que cambia el estado

```
(
dose = 0 ∧
  (
        (( r1>=r0) ∧ ( r2 = r1)) ∨
        (( r1 > r0) ∧  (r2<=r1)) ∨
        (( r1 < r0) ∧  ((r1-r2) > (r0-r1)))
  ) ∨
 dose = 4 ∧
  (
        (( r1<=r0) ∧ (r2=r1)) ∨
        (( r1 < r0) ∧ ((r1-r2)<=(r0-r1)))
  ) ∨
dose =(r2 -r1) * 4 ∧
  (
        (( r1<=r0) ∧  (r2 > r1)) ∨
        (( r1 > r0) ∧  ((r2 - r1)>=(r1 - r0)))
   )
)
capacity ' =capacity - dose
cumulative_dose' = cumulative_dose + dose
r0' = r1 ∧ r1' = r2
```

El programa:
- Compara el valor actual del nivel de azúcar con los dos anteriores
- si está subiendo, la bomba inyecta insulina
- mantiene total de insulina inyectada para controlar el máximo seguro

‘ aplicado a una variable refiere al valor luego de cambiar el estado

Esquemas de Salida

- Modelan lo que despliega el sistema:
- La pantalla muestra la dosis calculada y un mensaje de advertencia
- La alarma suena si el azúcar en la sangre es muy baja -debe ingerir azúcar

DISPLAY

ΔInsulin_Pump

---

display2!' = Nat_to_string (dose) $\wedge$
(reading? < 3 $\Rightarrow$ display1!'= "Azúcar baja" $\vee$
reading? > 30 $\Rightarrow$ display1!'= "Azúcar muy baja" $\vee$
reading? >=3 and reading?<=30 $\Rightarrow$ display1!'= "OK")

ALARM

ΔInsulin_Pump

(reading? < 3 $\vee$ reading? > 30) $\Rightarrow$ alarm!' = on $\vee$
(reading? >= 3 $\wedge$ reading?<=30) $\Rightarrow$ alarm!' = off