Unified Modeling Language 2

Autores:

Dr. Harald Störrle
MGM-EDV Beratung GmbH
y Universidad de Munich

Dr. Alexander Knapp
Universidad de Munich

Profesor:

Simon Pickin
Pablo Basanta Val
Ingeniería Telemática
Universidad Carlos III de Madrid

Versión abreviada y traduccion:

Simon Pickin, Angeles Manjarrés

Software de Comunicaciones 2007-2008

Para el tutorial original ver, por ejemplo:
`http://www.pst.ifi.lmu.de/veroeffentlichungen/UML 2.0-Tutorial.pdf`

Tutorial UML 2 – fuentes originales

- Esta presentación es una traducción de una versión abreviada de un tutorial redactado por A. Knapp & H. Störrle y presentado
  - en congresos internacionales (SEFM'06, VLHCC'06),
  - para formación en empresas (WEKA, MGM)
  - para formación universitaria (MNM).
- Los autores proporcionan formación y consultoría sobre UML y temas afines.
- Más artículos sobre UML véase:
  www.pst.ifi.lmu.de/~knapp
  www.pst.ifi.lmu.de/~stoerrle.

- Los diagramas de este tutorial se han tomado de los dos libros siguientes, ahora ampliamente usados para formación en Alemania.

ISBN 3-8273-2268-5

ISBN 3-8273-7143-0

Ejemplo utilizado a través del tutorial: Albatross Air Autopilot (AAA)

- Imagínese una línea aérea ficticia, *Albatross Air*, que está a punto de automatizar sus procesos en base a un nuevo sistema de información de gran alcance llamado “Autopiloto Albatross Air” abreviado a AAA.
- Las capacidades de AAA incluyen soporte para la reserva de vuelos, y para el *check-in* y embarque de pasajeros, así como un programa de puntos para clientes habituales llamado “Millas Albatross”.

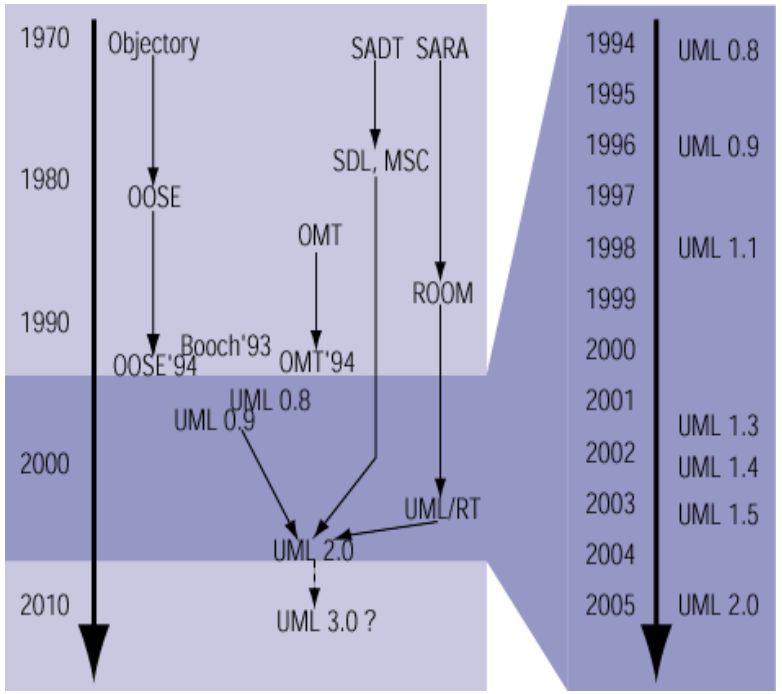
Unified Modeling Language 2

Parte 1 - Introducción

Software de Comunicaciones 2007-2008

1 - Introducción
Historia y predecesores

- El UML es la “lingua franca” de la ingeniería del software.
- Subsume, integra y consolida a la mayor parte de sus predecesores
- Tiene un alcance más amplio y un soporte mucho mejor (herramientas, libros, formación etc.) que otras notaciones.
- Transición UML1.x→UML 2:
  - números osos problemas resueltos,
  - muchos conceptos nuevos introducidos,
  - estructura interna revisada completamente y mejorada.
- Aunque UML 2 todavía tiene muchos problemas, es mucho mejor de lo que existía antes.

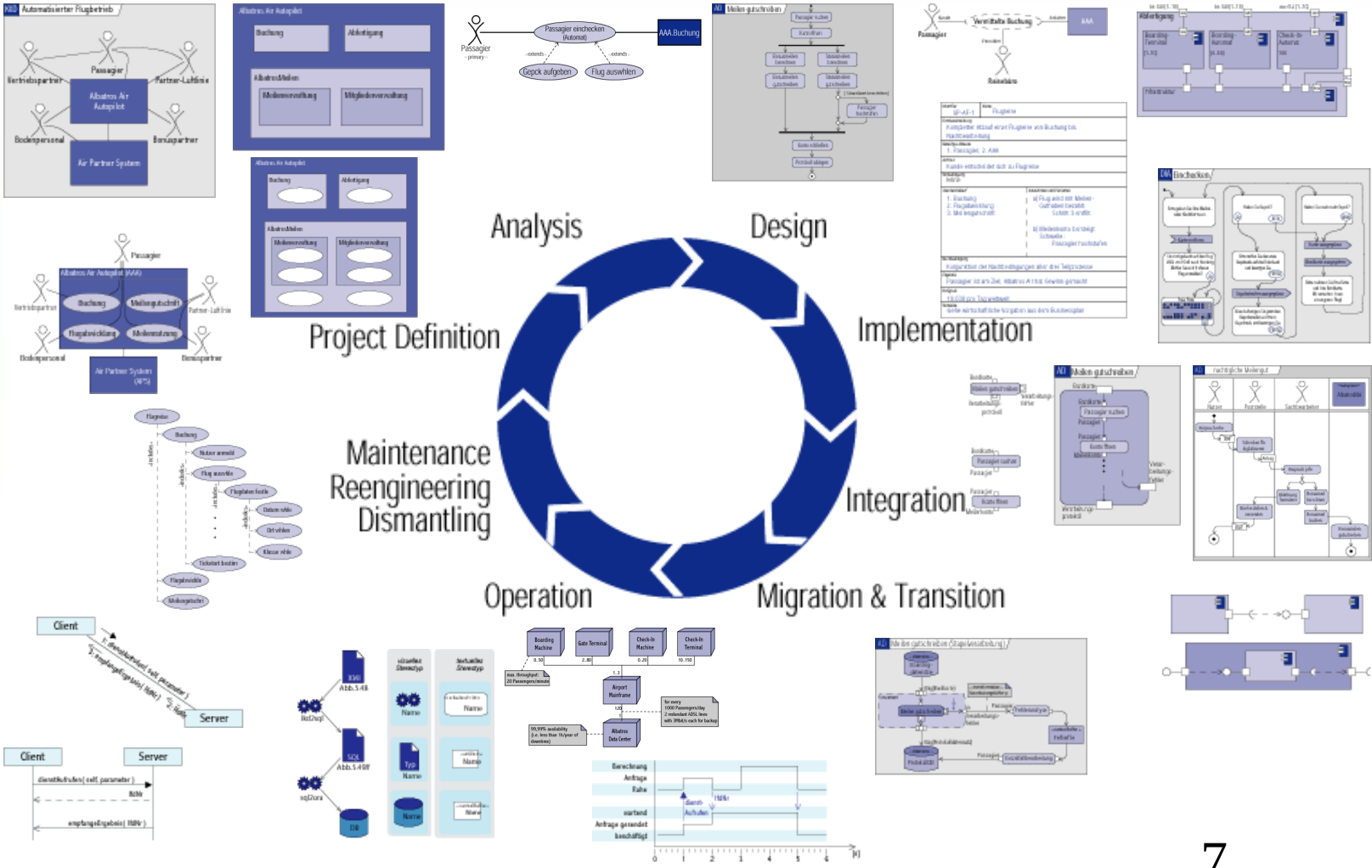


versión actual (“el estándar”)
2.1.1 del 5 de febrero de 2007

1 - Introducción
Escenarios de uso

- UML no se diseñó para usos específicos y limitados.
- Actualmente no hay consenso sobre su papel:
  - algunos ven UML sólo como una herramienta para esbozar diagramas de clase que representan programas Java.
  - otros creen que UML es “el prototipo de la próxima generación de lenguajes de programación”.
- En realidad, UML es una sistema de lenguajes (o notaciones, tipos de diagramas), cada uno de los cuales puede usarse en varias situaciones distintas.
- UML es aplicable, en mayor o menor grado, para múltiples fines y durante todas las fases del ciclo de vida del software.

1 - Introducción
Escenarios de uso



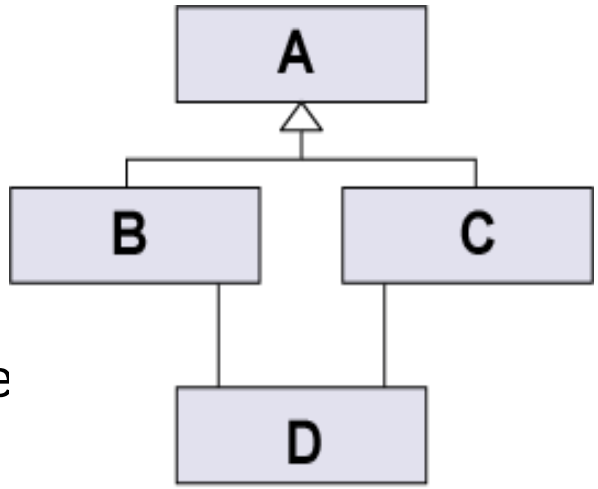
1 - Introducción
Tipos de diagramas de UML 2

- UML es un sistema coherente de lenguajes más que un lenguaje único.
- Cada lenguaje tiene su enfoque particular.

Structure	Class Diagram		static structure (generic/snapshot)
	Composite Structure Diagram		logical system structure
	Component Diagram		physical system structure
	Deployment Diagram		computing infrastructure / deployment
	Package Diagram		containment hierarchy
Behavior	Use Case Diagram		abstract functionality
	Activity Diagram		controlflow and dataflow
	Interaction	Sequence Diagram	*interactions by message exchange*	message exchange over time
		Communication Diagram		structure of interacting elements
		Timing Diagram		coordinated state change over time
		Interaction Overview Diagram		flows of interactions
	State Machine Diagram			event-triggered state change

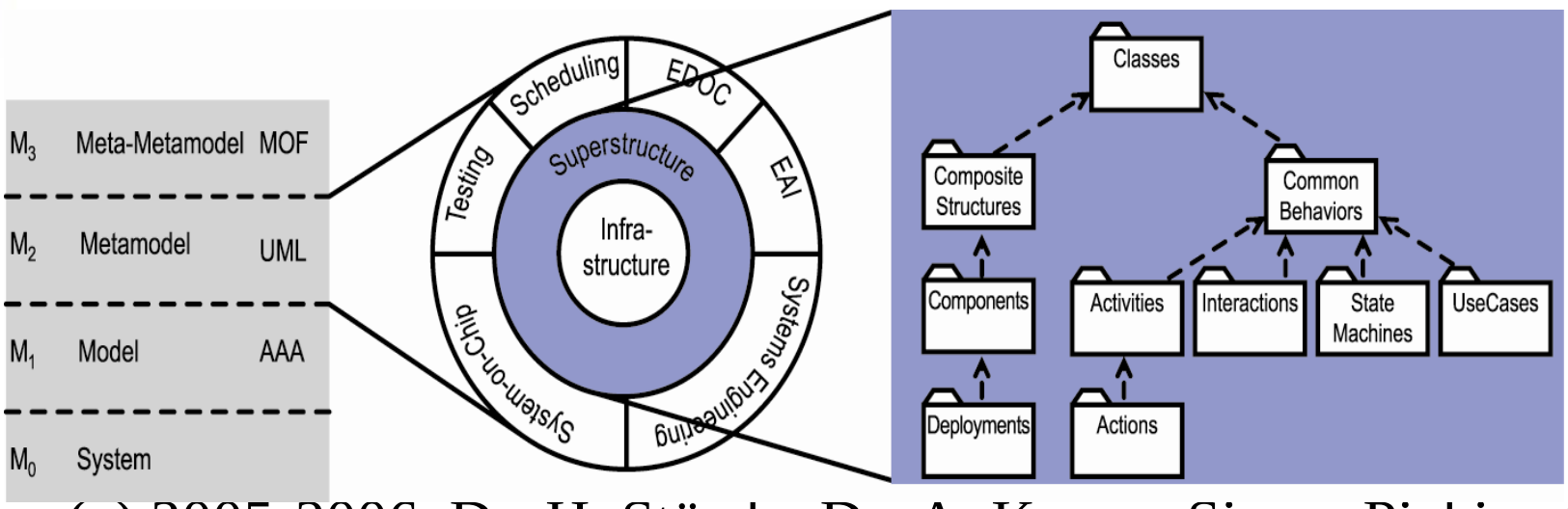
1 - Introducción
Los tipos de diagramas también dependen de su uso

- Cada tipo de diagrama puede usarse en multitud de contextos, para cada uno de los cuales sean pertinentes diferentes reglas y buenas prácticas.
- Por ejemplo, los diagramas de clase pueden usarse tanto durante el análisis como durante la implementación.
- Durante el análisis, este diagrama de clases es malo, o al menos dudoso.
- Durante la implementación, es malo si y solo si no se corresponde con el código (u otra estructura) para cuya representación es usado.

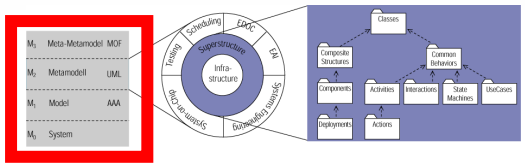


1 - Introducción
Estructura Interna: Visión de conjunto

- UML se estructura mediante un enfoque de metamodelado de cuatro capas.
- La capa M_2 se denomina metamodelo.
- El metamodelo se estructura a su vez en anillos, en uno de los cuales, denominado superestructura, se definen los conceptos ("el metamodelo" propiamente dicho).
- La Superestructura se estructura en un árbol de paquetes

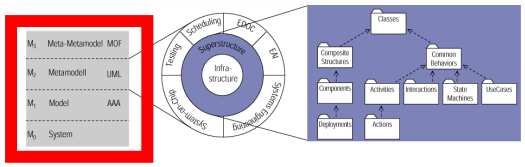


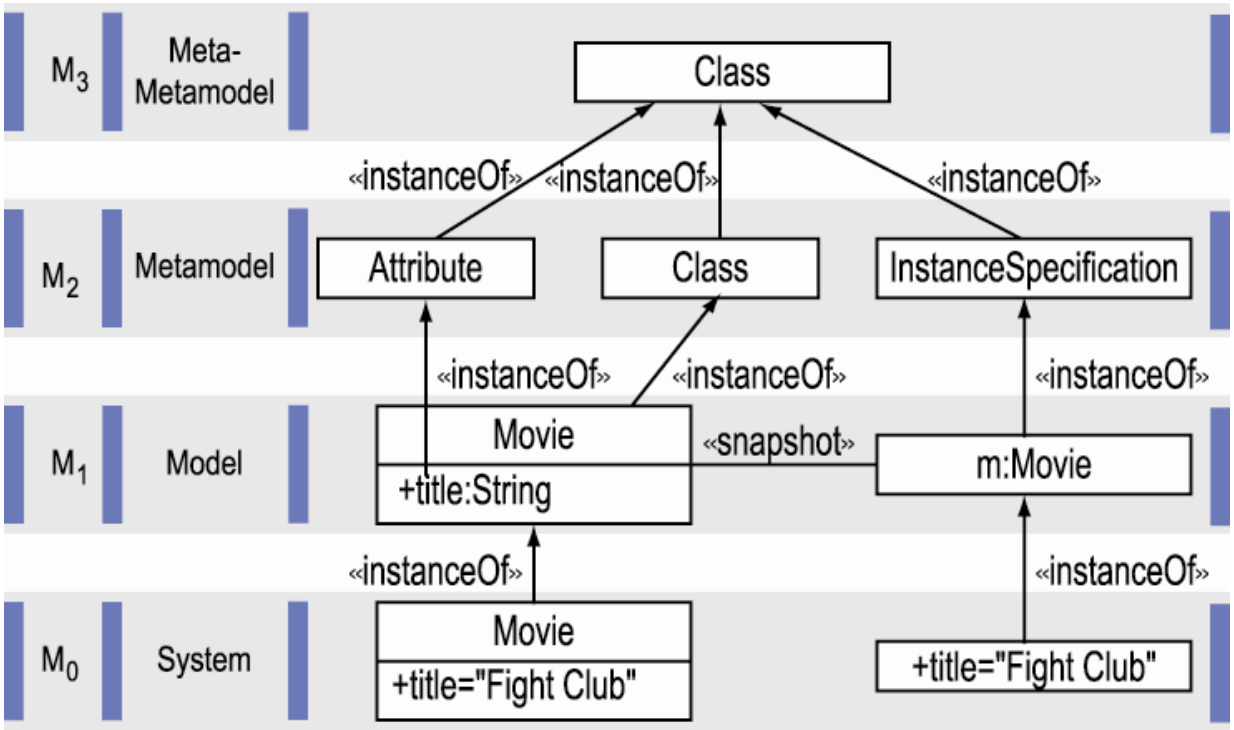
1 - Introducción
Estructura Interna: Niveles



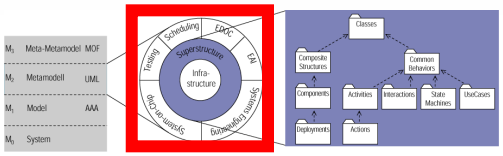
M_3	Meta-Metamodel	EBNF	Meta Object Facility (MOF)
M_2	Metamodel	Java grammar	Unified Modeling Language (UML) Common Warehouse Metamodel (CWM)
M_1	Model	a Java program	Albatros Air Autopilot
M_0	System	an execution of a Java program	a runtime state in a deployment of Albatros Air Autopilot

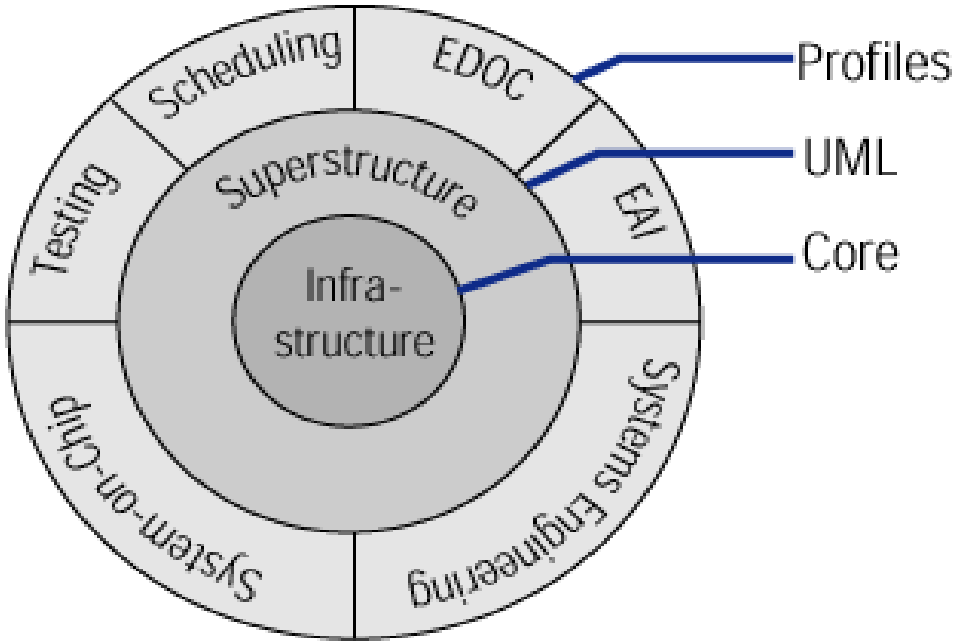
1 - Introducción
Estructura Interna: Capas





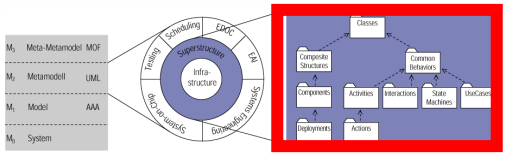
1 – Introducción
Estructura Interna: Anillos

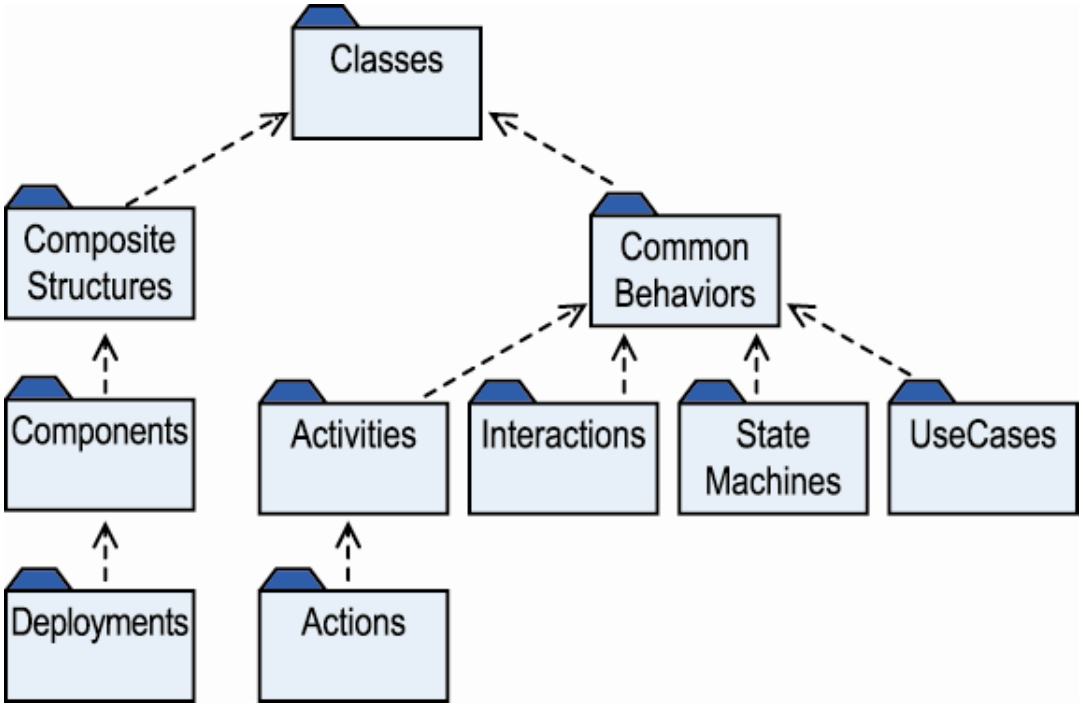




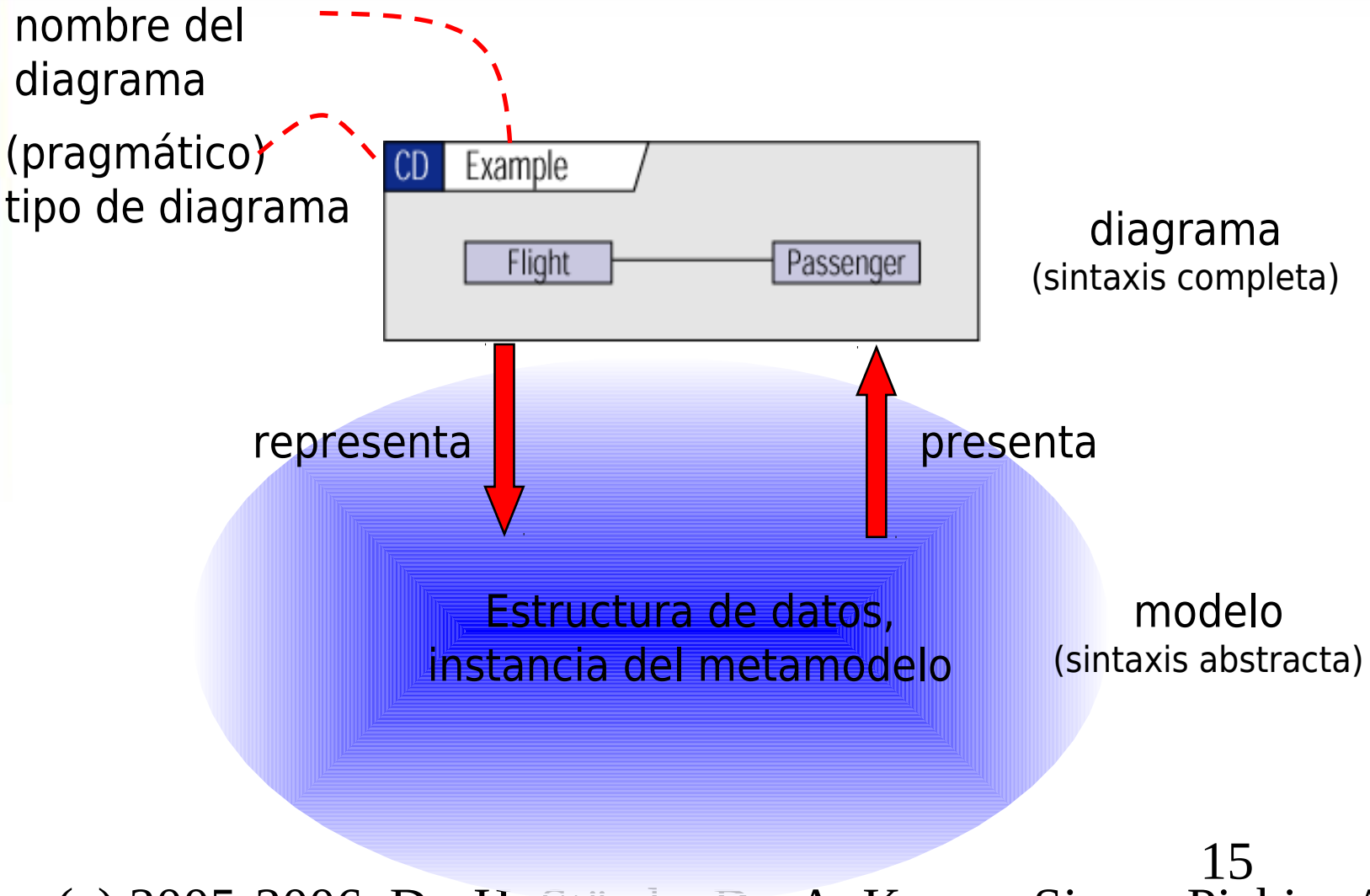
Software de Comunicaciones 2007-2008

1 - Introducción
Estructura Interna: Paquetes





1 - Introducción
Diagramas y Modelos



1 - Introducción
UML no es (sólo) orientado a objetos

- Un popular concepto erróneo del UML lo considera obligadamente orientado a objetos.
- Es cierto que
  - UML define conceptos como clase y generalización;
  - UML se define (principalmente) mediante un conjunto de modelos de clases;
  - UML2 redescubre la idea de comportamiento incorporado en objetos.
- Sin embargo, UML 2
  - también abarca muchos otros conceptos de origen pre – o no-OO (Actividades, Máquinas de Estados, Interacciones,…);
  - puede utilizarse en proyectos de desarrollo con independencia completa de sus lenguajes de implementación;
  - no está vinculado a ningún lenguaje ni paradigma de lenguajes, ni por accidente ni a propósito.

1 - Introducción
UML 1.x vs. UML 2

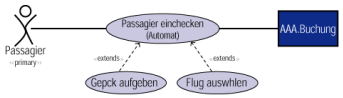
- UML 2 tiene varias ventajas frente a UML 1.x:
  - muchos nuevos conceptos potentes
  - definiciones mucho mejores (esto es, semántica)
  - estructuración interna mejorada
- Sin embargo, aunque UML 2 está mucho mejor definido que UML 1.5, su estado no es aún satisfactorio, p.e..
  - sintaxis
    - notación sobrecargada: demasiados sinónimos,demasiado azucarado,
    - falta de ortogonalidad notacional, hay quien ni siquiera desea esto,
  - semántica
    - falta de semántica precisa: informal, dfns. contradictorias y poco claras,
  - pragmática
    - falta de base metodológica tal como condiciones de coherencia de modelos, tipos de uso etc.
- Aun así, es el lenguaje de modelado más completo (“unificado”)

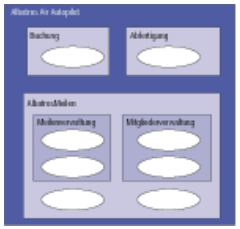
1 - Introducción
Conclusiones

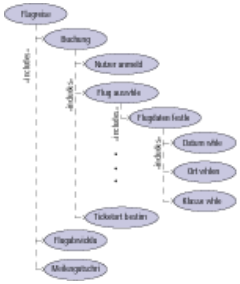
- UML is la lingua franca de la ingeniería del software.
- Tiene muchos problemas, pero aun así es lo mejor hasta la fecha.
- Puede usarse durante todo el ciclo de desarrollo soft.
  - para planear, analizar, diseñar, implementar y documentar
- El UML está estructurado
  - mediante un enfoque de metamodelado de 4 niveles
    (M_0: sistema, M_1: modelo, M_2: metamod., M_3: metametamod.),
  - el metamodelo se estructura en 3 anillos
    (infraestructura, superestructura, extensiones),
  - la superestructura se organiza como un árbol de paquetes.
    (e.g. Acciones, Actividades, Comportamientos Comunes, Clases)
- UML no es obligadamente orientado a objetos.

Unified Modeling Language 2

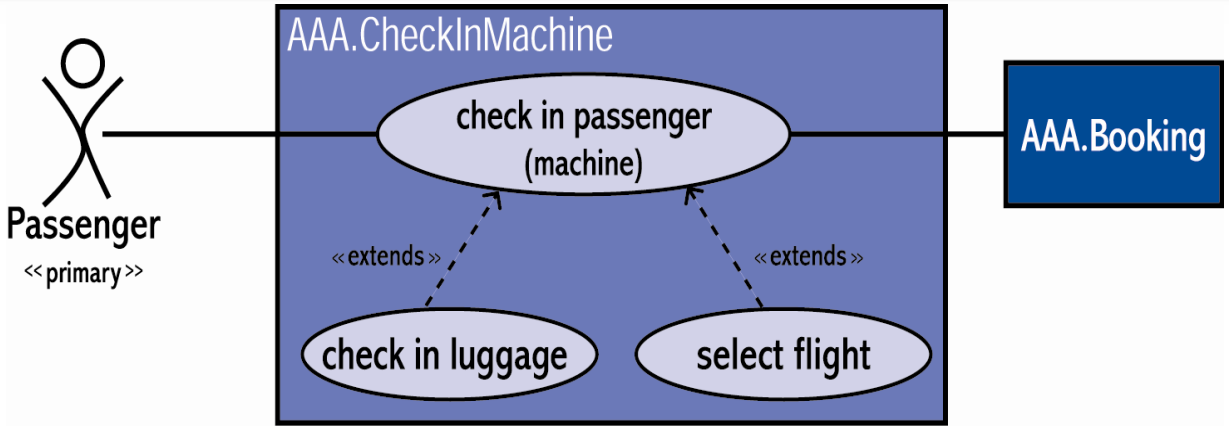
Parte 2 – Casos de Uso







2 – Casos de Uso
Un primer vistazo



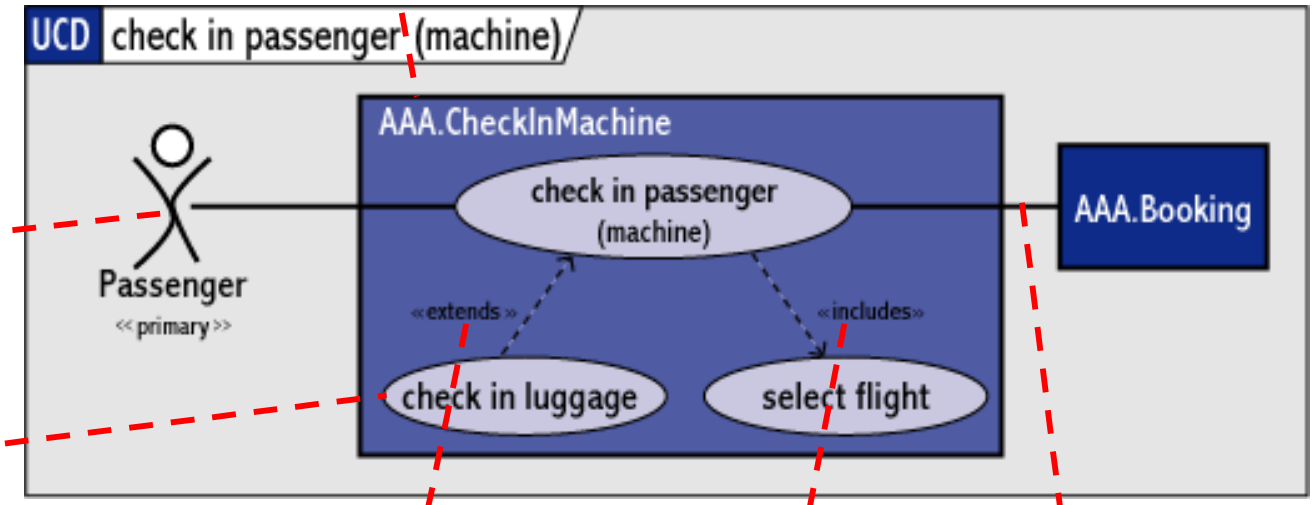
- Aspectos mostrados
  - límites y contexto del sistema
  - sistemas usuarios y vecinos
  - funcionalidades
  - relaciones entre funcionalidades (llamada/dependencia, taxonomía)
  - requisitos funcionales
  - algunos requisitos no funcionales (“calidad”) como comentarios/ anotaciones

2 – Casos de Uso
Escenarios de Uso

- Inventario de casos de uso/ arquitectura del dominio
  - catálogo completo de todos los subdominios y (grupos de) procesos de negocio y funciones de negocio
  - visión de conjunto de las capacidades (del dominio) del sistema
- Casos de uso “clásicos”
  - ilustran el contexto de la funcionalidad individual
  - útil en el diseño/documentación de los procesos de negocio (esto es, en la fase de análisis y en la reingeniería)
- Tabla de casos de uso / casos de test
  - descripción detallada esquemática de procesos/funciones/casos de pruebas
- Árbol de funciones
  - descomposición funcional del comportamiento del sistema
  - útil para construcción no-OO y re-arquitectura de sistemas pre-OO

2 – Casos de Uso
Conceptos principales (sintaxis concreta)

Clase (también posible: **Componente**)



Actor

Caso de Uso

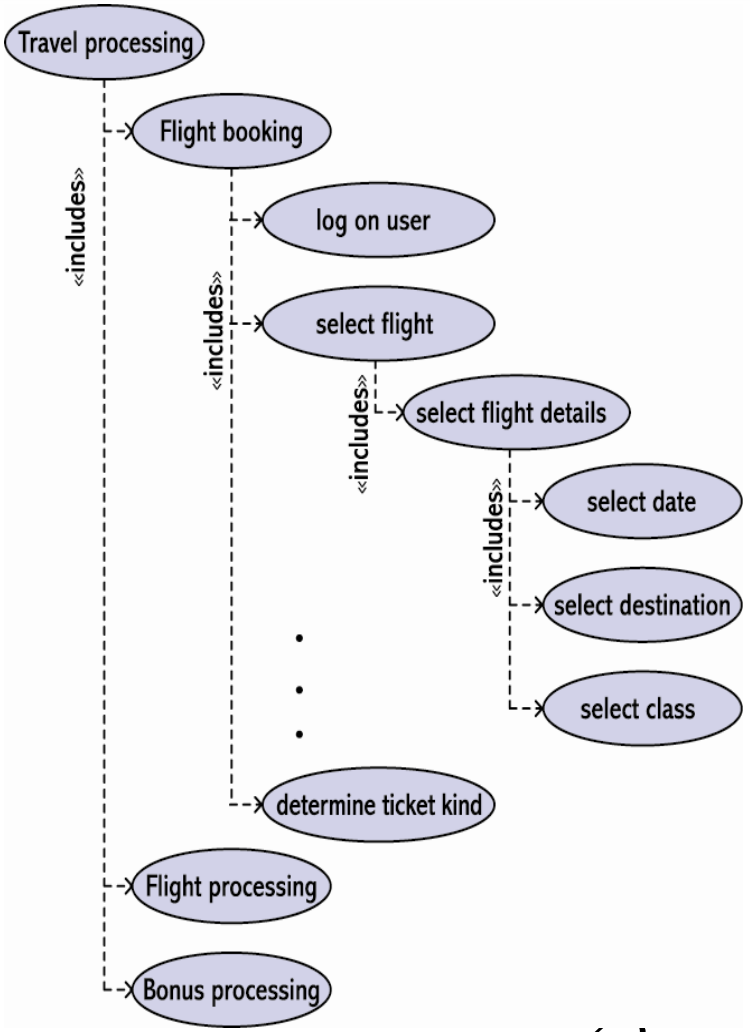
extiende (una **Dependencia**)

incluye (una **Dependencia**)

Asociación

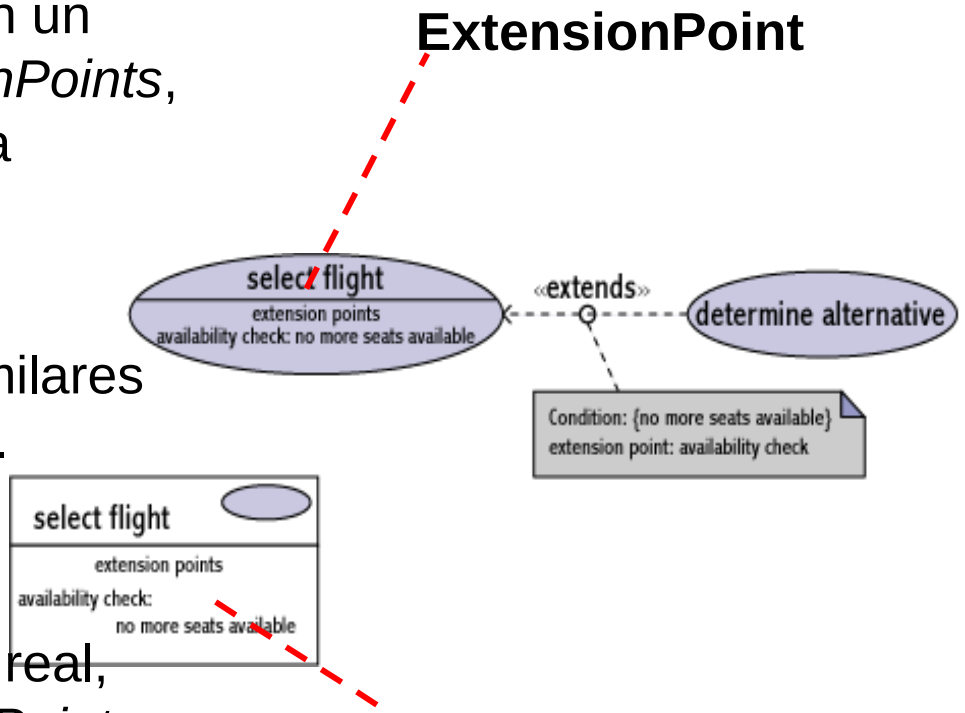
2 – Casos de Uso
Inclusión & extensión

- Inclusión
  - la llamada común
  - dirigida desde el llamante al llamado
  - puede tener lugar una o más veces
- Extensión
  - cubre comportamiento variante o excepcional
  - la relación se dirige desde la excepción al caso estándar
  - puede o no tener lugar
  - ocurre como máximo una vez



2 – Casos de uso
Puntos de extensión

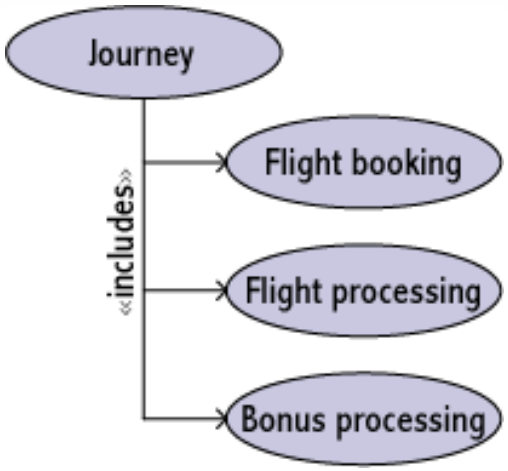
- Una extensión ocurre en un (denominado) *ExtensionPoints*, cuando se satisface una condición específica.
- En cierto modo los ExtensionPoints son similares a los *user exits* o *hooks*.
- En sistemas del mundo real, hay **muchos** *ExtensionPoints*, muchos de los cuales están pobremente documentados

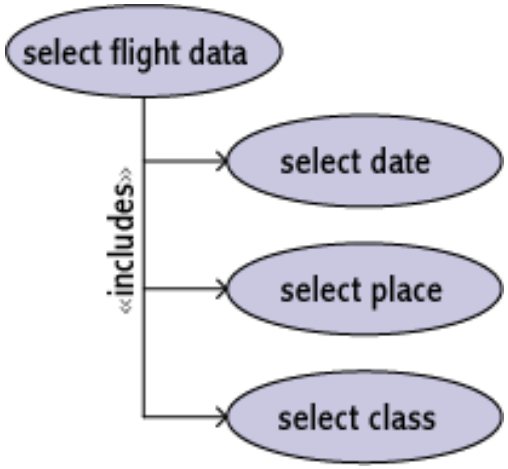


UseCase con *ExtensionPoints*, sintaxis alternativa adecuada para un gran número de *ExtensionPoints*

2 – Casos de Uso
Cualquier nivel de abastracción es válido

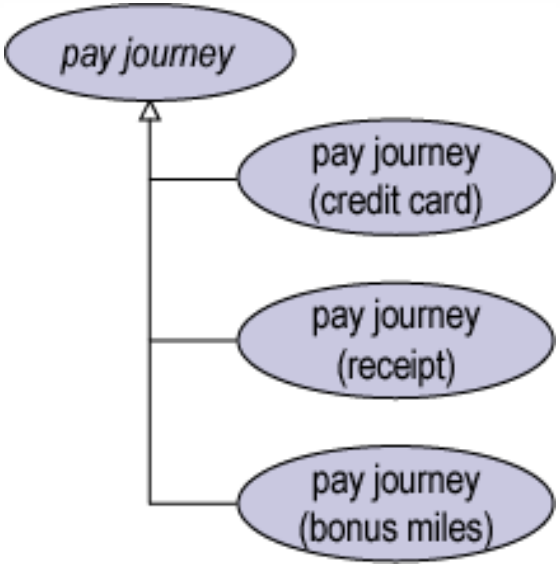
- Un caso de uso representa una funcionalidad individual de un sistema.
- Existen sistemas en cualquier nivel de granularidad.
- Así, los casos de uso pueden utilizarse para funcionalidades de cualquier granularidad :
  - desde procesos de negocio de alto nivel,
  - via servicios (web),
  - a métodos individuales de funciones.

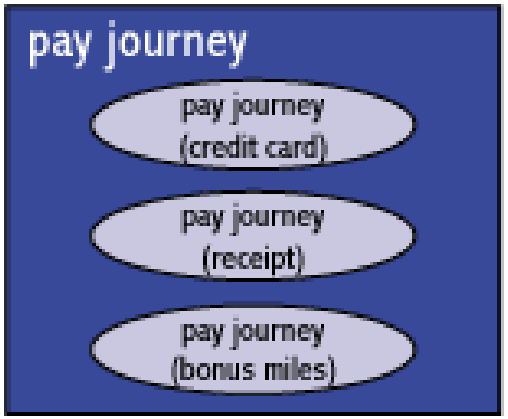




2 – Casos de Uso
Generalización

- Al igual que todos los *Classifiers*, los casos de uso pueden organizarse en jerarquías taxonómicas.
- Esto es particularmente útil para catálogos de funcionalidades.
- Desde el punto de vista metodológico, los casos de uso abstractos son similares a los subsistemas funcionales.





2 – Casos de Uso
Conclusiones

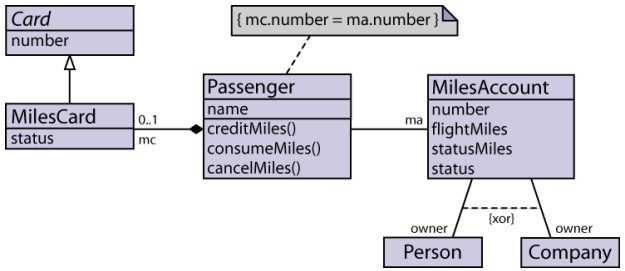
- Los Casos de Uso pueden utilizarse para representar una visión de alto nivel de la funcionalidad, como
  - visión de conjunto de la funcionalidad / arquitectura del dominio
  - descripción detallada del contexto del caso de uso individual
  - árbol de funciones (particularmente para reingeniería y propósitos de documentación)

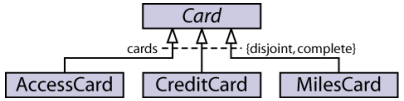
- UML aún no incluye un esquema textual para describir casos de uso.

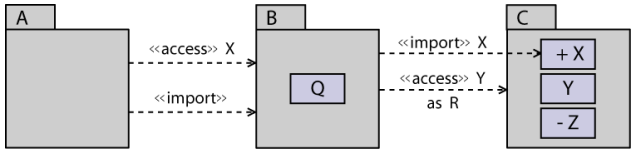
- Basicamente, los casos de uso de UML 2 son iguales a los de UML 1.x.

Unified Modeling Language 2

Parte 3 – Clases y paquetes



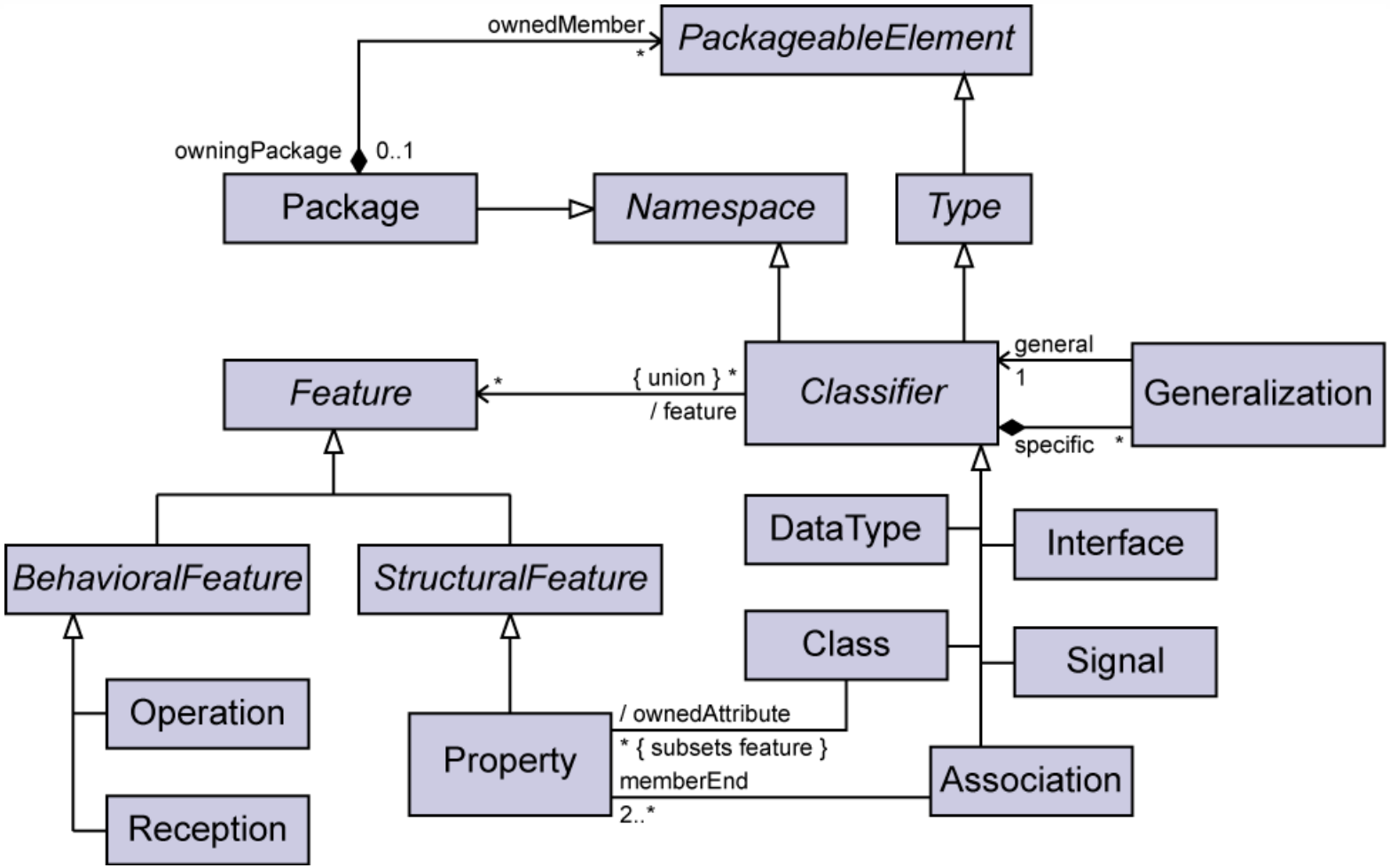




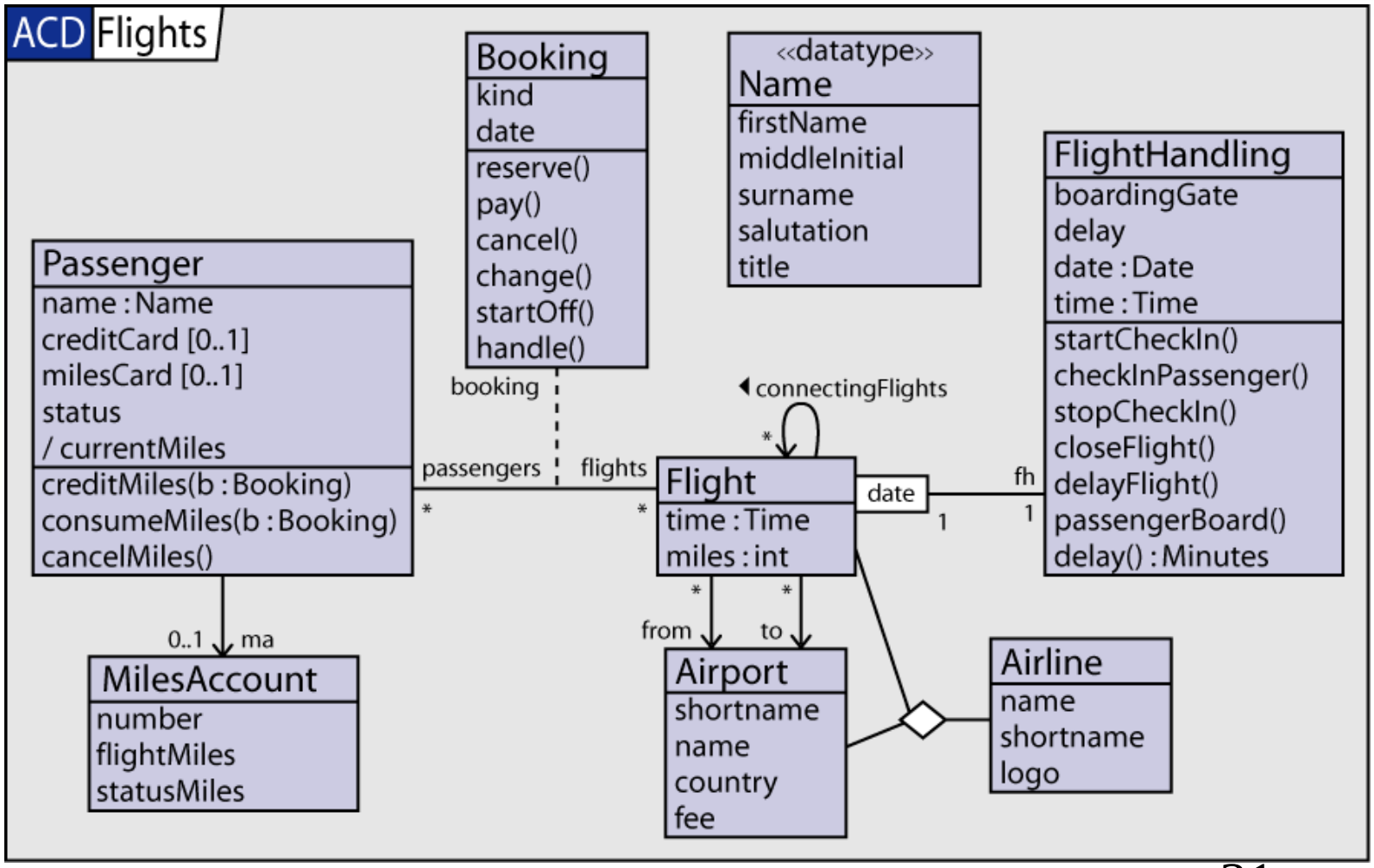
3 – Clases y paquetes
Escenarios de uso

- Las clases y sus relaciones describen el vocab. de sistema.
  - **análisis**: ontología, taxonomía, diccionario de datos, …
  - **diseño**: estructura estática, patrones, …
  - **implementación**: contenedores de código, tablas de bases de datos, …
- Las clases pueden utilizarse con distinto sentido en distintas fases del desarrollo software.

3 – Clases y paquetes
Metamodelo

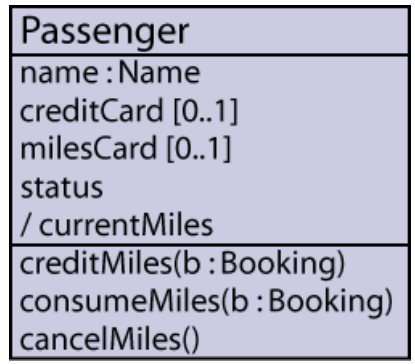


3 – Classes and packages
Ejemplo 1: diagrama de clases de analisis



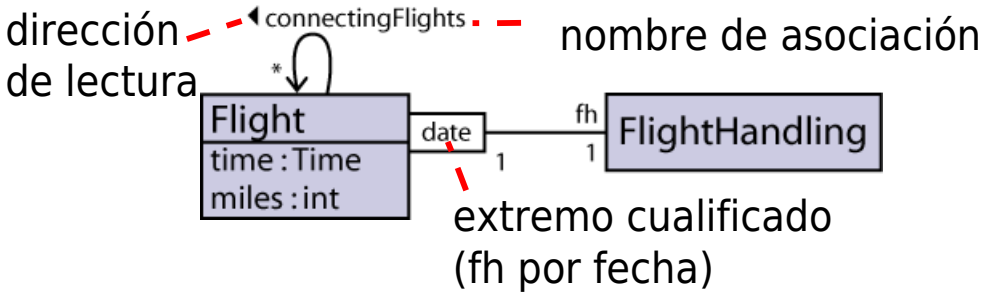
3 – Clases y paquetes
Clases

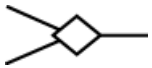
- Las clases describen un conjunto de instancias con *features* (y semántica) comunes
  - clases inducen tipos (que representan conjuntos de valores).
  - son espacios de nombres (que contienen elem. nombrados).
- *Features* estructurales
  - propiedades
    - comúnmente conocidas como atributos
    - describen estructura/estado de instancias
    - pueden tener multiplicidades
      (por defecto: 1 para extremos de asociaciones,
      0..* = * para otros *features*)
- *Features* comportamentales
  - operaciones
    - servicios que pueden invocarse
    - no hace falta que se respalden por un método



3 – Clases y paquetes
Asociaciones

- … describen conjuntos de tuplas cuyos valores se refieren a instancias tipadas
  - en particular, relaciones estructurales entre clases
  - las instancias de asociaciones se denominan enlaces.

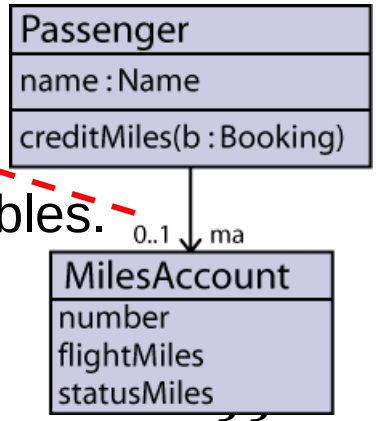




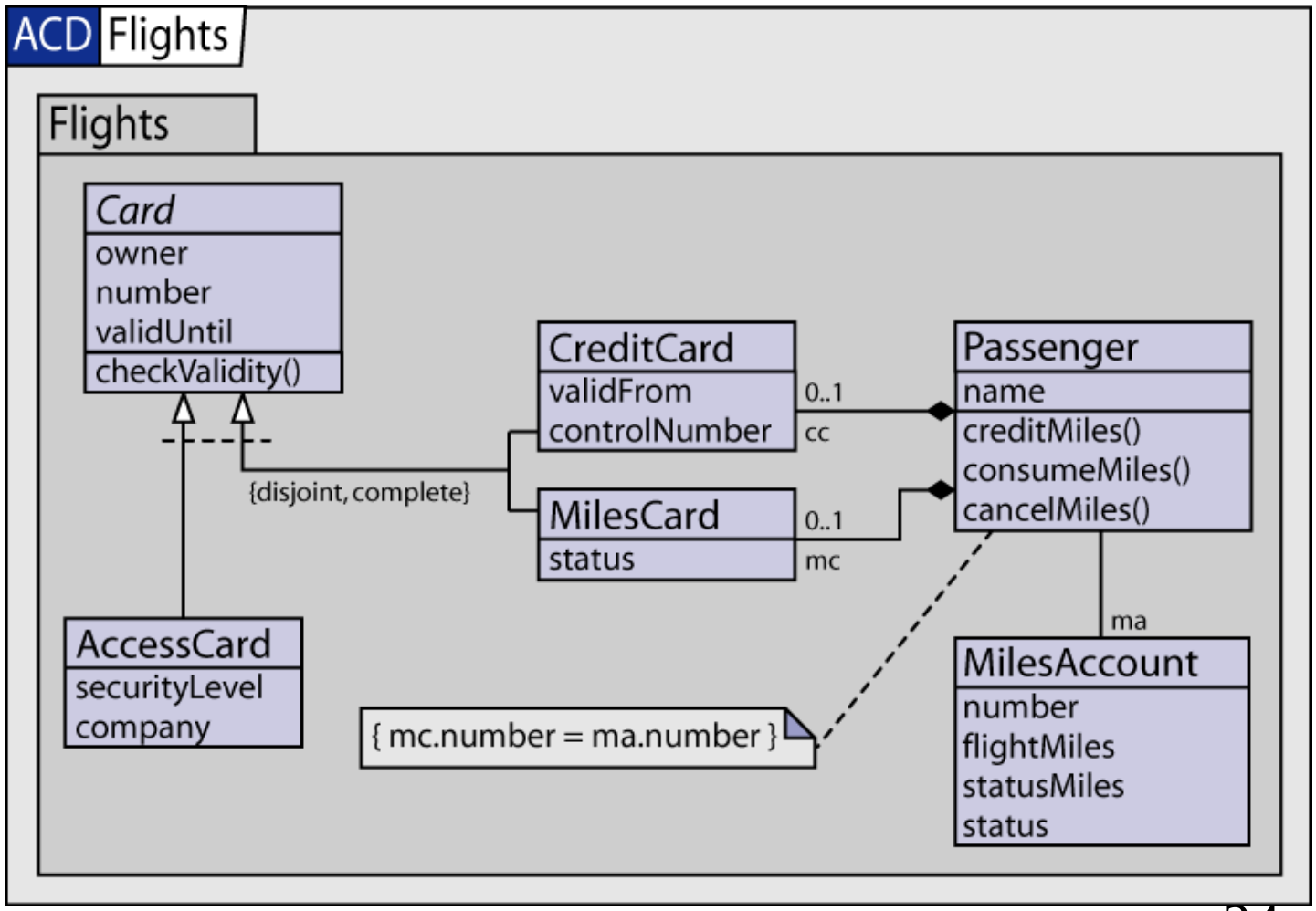
asociación ternaria

- Los extremos de asociación son propiedades.
  - corresponden a propiedades de la clase opuesta (pero la multiplicidad por defecto es 1)
- Los extremos de asociación pueden ser navegables.
  - en contraste con las propiedades generales

navegable no navegable



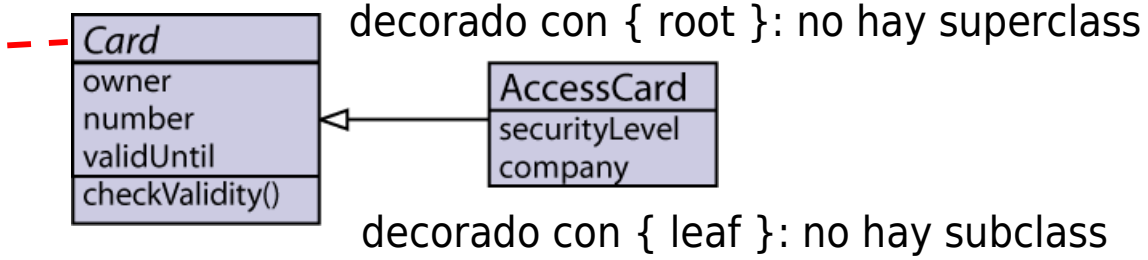
3 – Clases y paquetes
Ejemplo 2: diagrama de clases de análisis



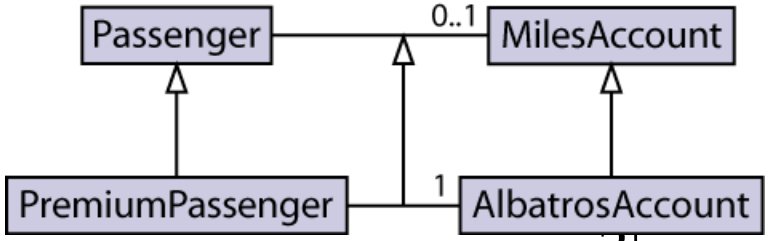
3 – Clases y paquetes
Herencia

- Generalización
  - relaciona una clase específica con una clase más general
  - las instancias de la clase específica lo son de la general
  - los *features* de la clase general están especificados implícitamente para la clase específica

{ abstract } class (sin instancias directas, sólo las especializaciones pueden tenerlas)

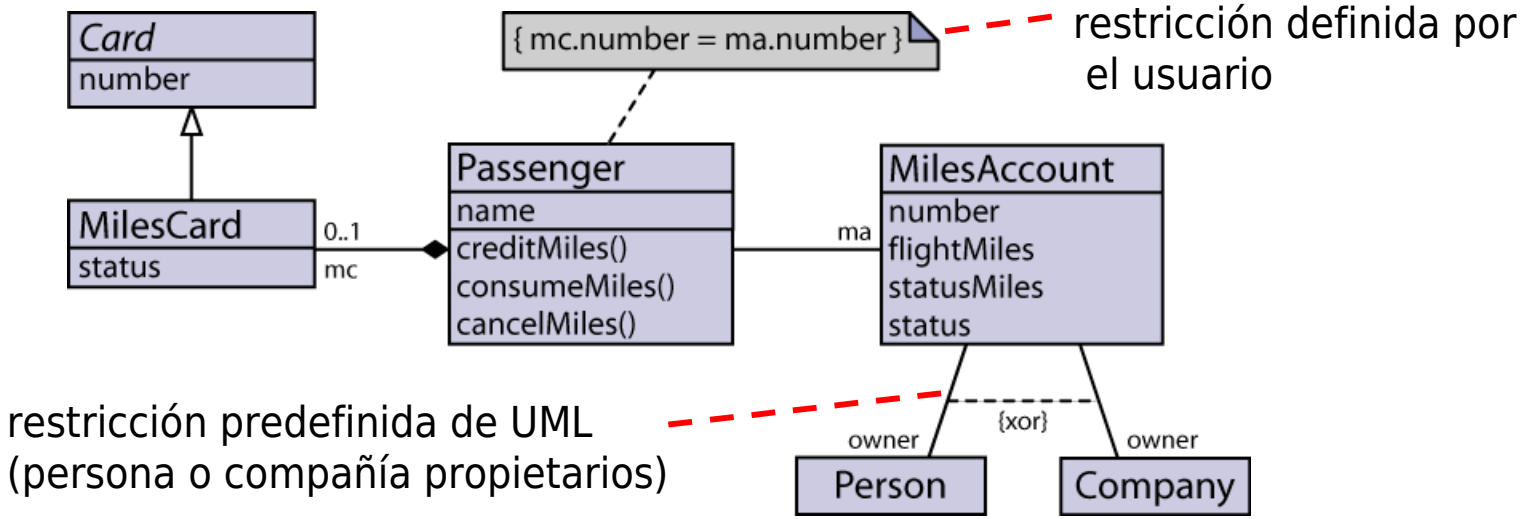


- También se aplican a las asociaciones
  - puesto que son *Classifiers*



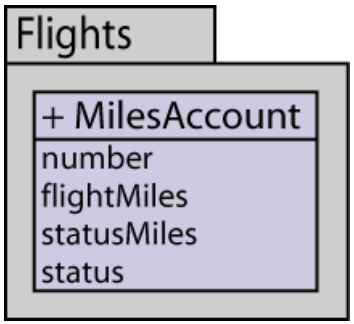
3 – Clases y Paquetes
Restricciones

- Restringen la semántica de los elementos del modelo.
  - pueden aplicarse a uno o más elementos
  - no se prescribe ningún lenguaje
    - OCL se usa en la especificación de UML 2
    - también puede usarse lenguaje natural

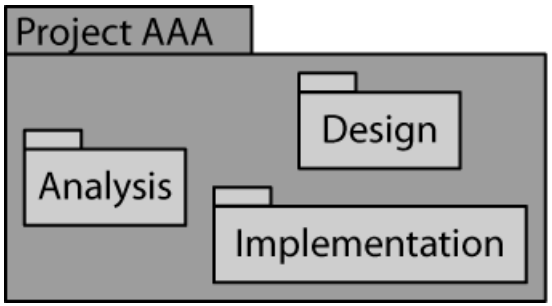


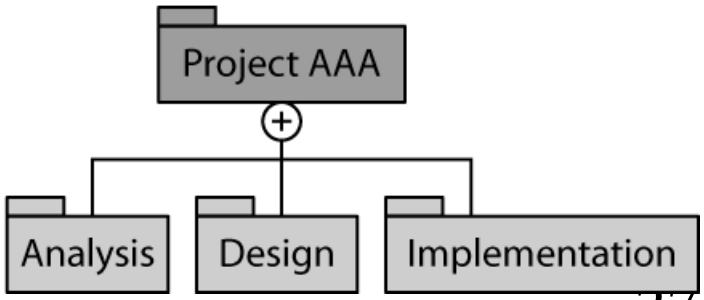
3 – Clases y paquetes
Paquetes (1)

- Agrupan elementos.
  - proporcionan un espacio de nombre para estos elementos.
  - los elementos de un paquete pueden ser
    - public (+, visibls desde fuera; por defecto)
    - private (-, no visibles desde fuera)
  - acceso a elementos públicos mediante nombres cualificados
    - e.g., Flights::MilesAccount



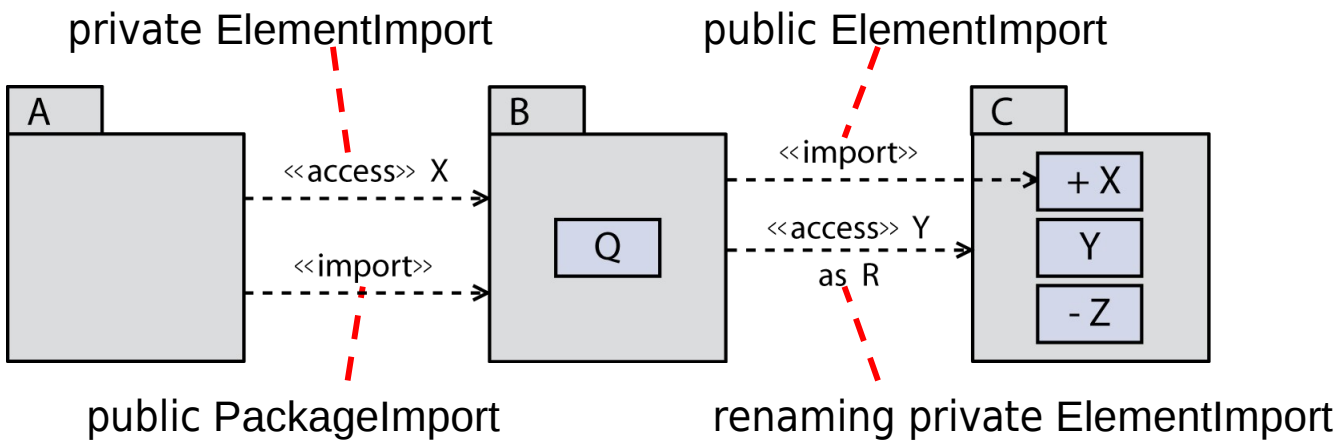
Variantes notacionales





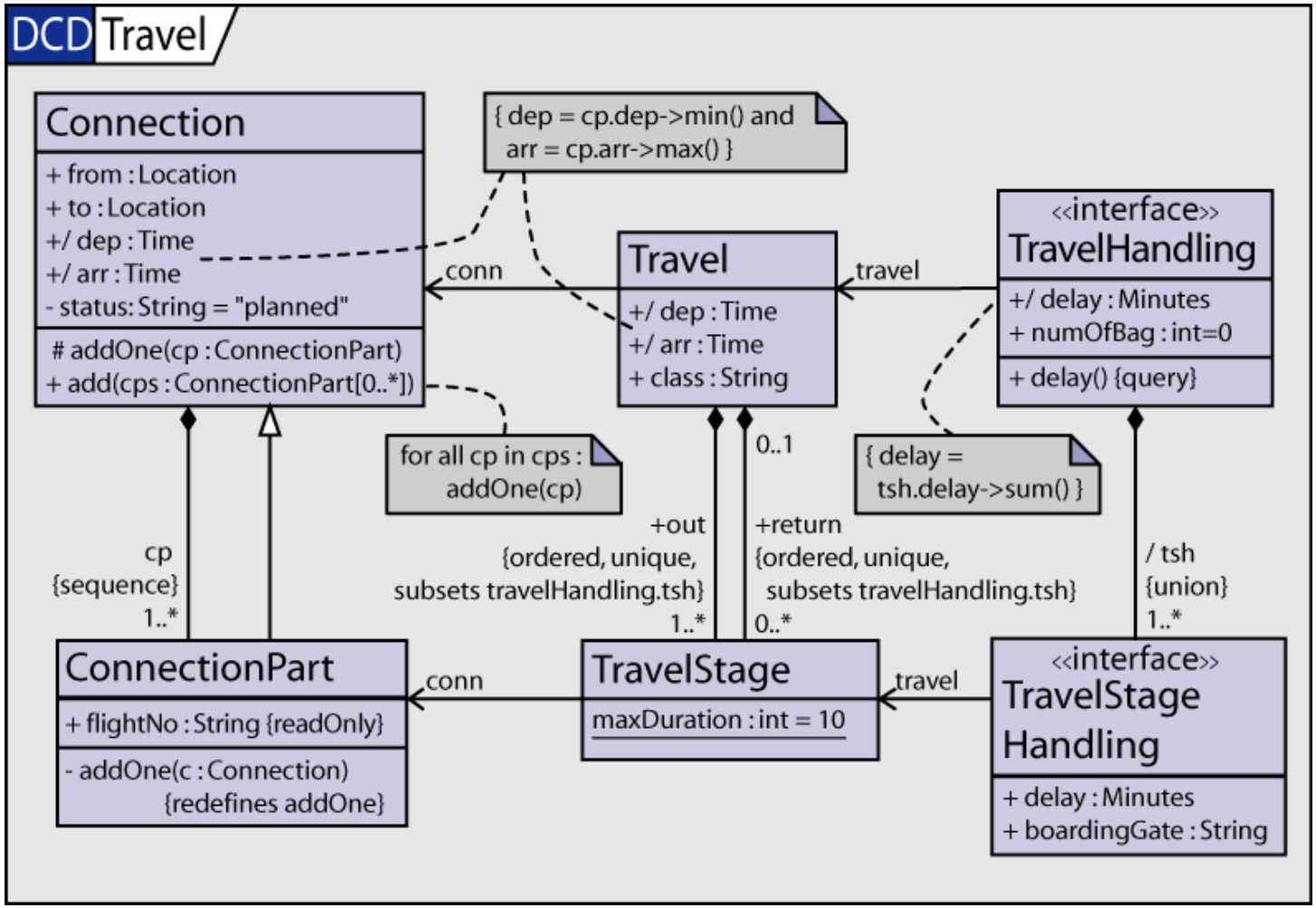
3 – Clases y paquetes
Paquetes (2)

- Importan nombres cualificados simplificados.



Package	Element	Visibility	
A	**X**	**private**	***import* de elemento privado específico (de otro modo public reescribe private)**
A	**Q**	**public**	**los restantes elementos visibles de B**
B	**X**	**public**	***import* público**
B	**Q**	**public**	**visibilidad por defecto**
B	**R**	**private**	***import* privado con renombramiento**

3 – Clases y paquetes
Ejemplo 3: diagrama de clases de diseño



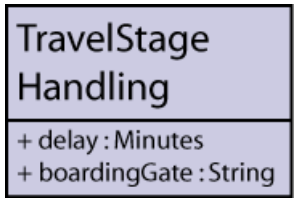
3 – Clases y paquetes
Features (1)

- Tipos de visibilidad (no hay tipo por defecto)

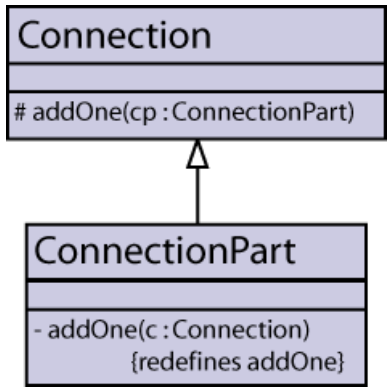
		visible para elementos …
+	**public**	**que pueden acceder al espacio de nombres a que pertenece (por pertenencia suya, mediante importación, o mediante acceso)**
#	**protected**	**con generalización al espacio de nombres a que pertenece**
~	**package**	**en el mismo paquete que el espacio de nombres a que pertenece**
-	**private**	**sólo en el espacio de nombres a que pertenece**

3 – Clases y paquetes features (2)

- … pertenecen a un espacio de nombres (p.e., clase o paquete)



- … son redefinibles (a menos que se decoren con { leaf })
  - en clases que especializan a la clase contexto



- … pueden definirse en el nivel de instancia o de clase

3 – Clases y paquetes
Propiedades (de propiedades) (1)

{ ordered }	{ unique }	Tipo de colección
√	√	`OrderedSet`
√	×	`Sequence`
×	√	`Set` (default)
×	×	`Bag`

/ ({ derived }) — **computable a partir de otra información (defecto: false)**

{ readOnly } — **pueden leerse, no escribirse (defecto: false = unrestricted)**

{ union } — **union de propiedades de subconjuntos (implica *derived*)**

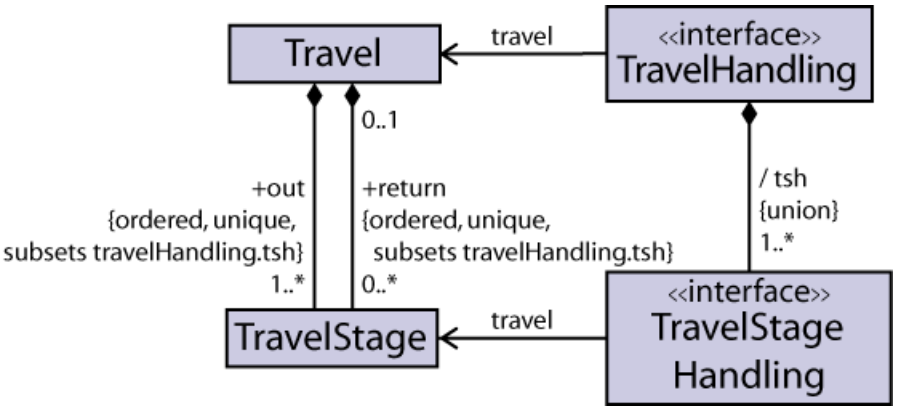
{ subsets ...} — **de qué propiedad esta propiedad es subconjunto**

3 – Clases y paquetes
Propiedades (de propiedades) (2)

- Tipos de agregación (defecto: *none*)

none	———	**reference**
shared	◇———	undefined (!)
composite	◆———	**value**

- Ejemplo de propiedades…

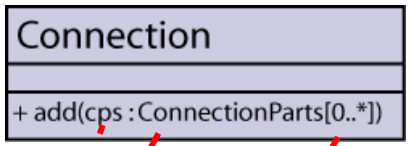


3 – Clases y paquetes
Características comportamentales

- … realizadas por comportamientos (p.e., código, máquina de estados).
  - { abstract }: (virtual) los features comportamentales no declaran comportamiento
    - las especializaciones deben proporcionar el comportamiento
  - pueden declararse las excepciones que pueden ser lanzadas

3 – Clases y paquetes
Operaciones

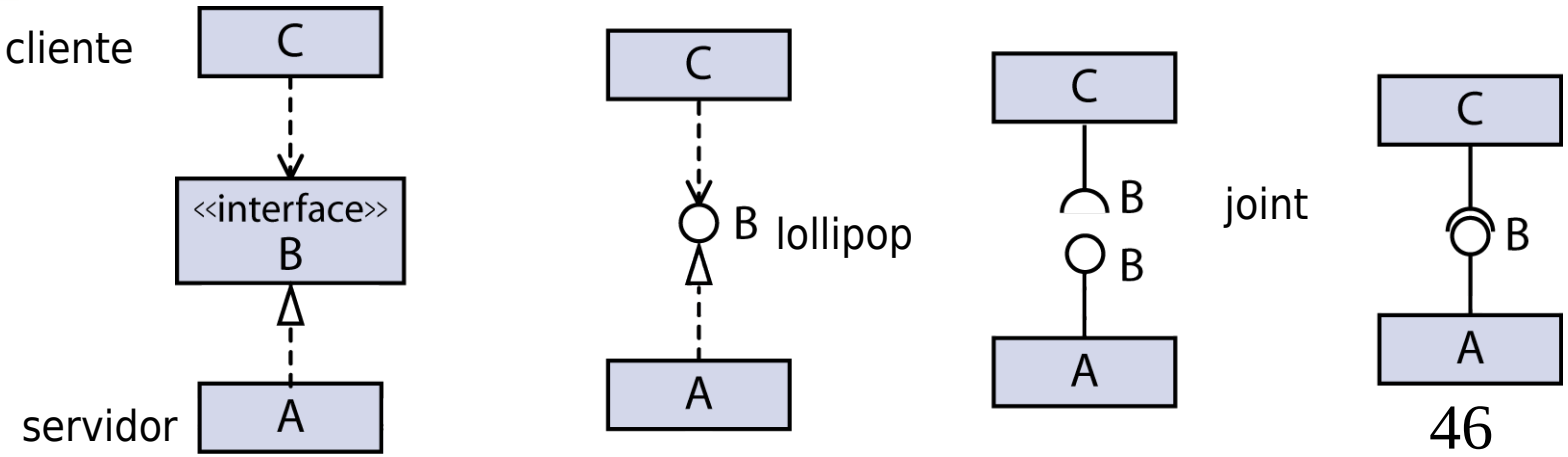
- Una operación especifica el nombre, tipo devuelto, parámetros formales,y restricciones para invocar un comportamiento asociado.
  - «pre» / «post »
    - las precondiciones restringen el estado del sistema al invocar la operación
    - las postcondiciones restringen el estado del sistema cuando se ha completado la operación
  - { query }: la invocación no tiene efectos laterales
    - «body»: la condición body describe los valores devueltos
  - { ordered, unique } como para las propiedades, pero para los valores devueltos
  - se pueden declarar excepciones



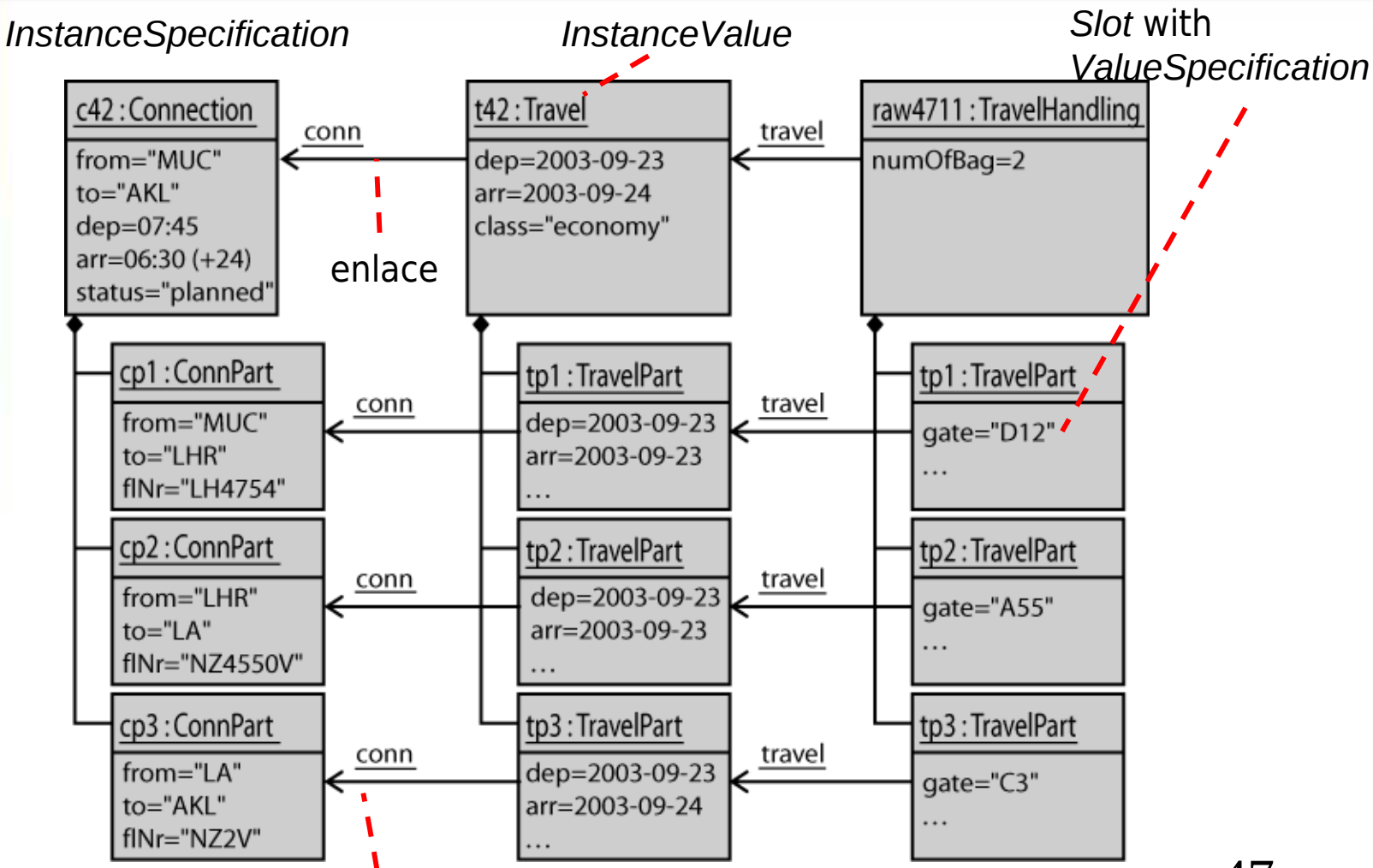
3 – Clases y paquetes
Interfaces

- Declaran un conjunto de *features* públicas y obligaciones coherentes.
  - es decir, especifican un contrato para los implementadores (realizadores)

Existen varias notaciones para la relación cliente/servidor



3 – Clases y paquetes
Diagrama de objetos

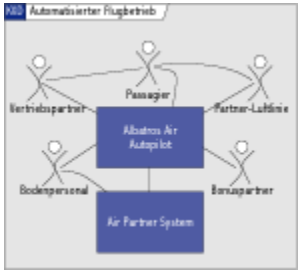


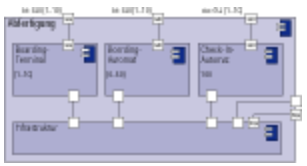
3 – Clases y paquetes
Conclusiones

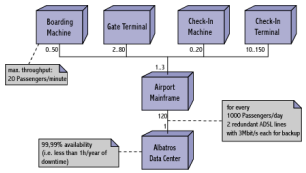
- Clases y sus relaciones describen el vocabulario de un sistema.
- Clases describen conjuntos de instancias con *features*
  - StructuralFeatures (Propiedades)
  - BehavioralFeatures (Operaciones, Recepciones)
- Las asociaciones describen relaciones estructurales entre clases
  - los extremos de asociación son propiedades.
- Las generalizaciones relacionan clases con clases más generales.
- Los paquetes agrupan elementos
  - y proporcionan un *Namespace* para elementos agrupados.
- *InstanceSpecifications* y enlaces describen fotogramas del

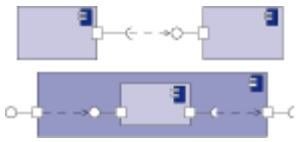
Unified Modeling Language 2

Part 4 - Arquitectura



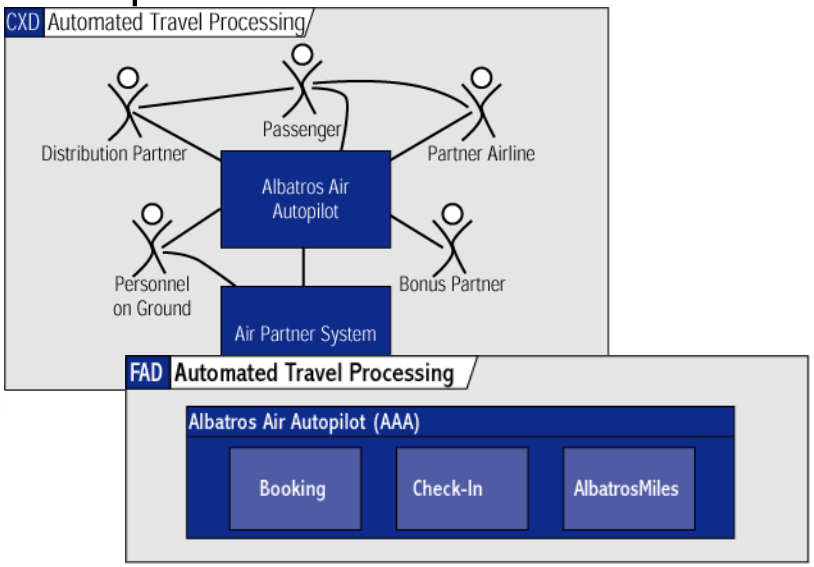




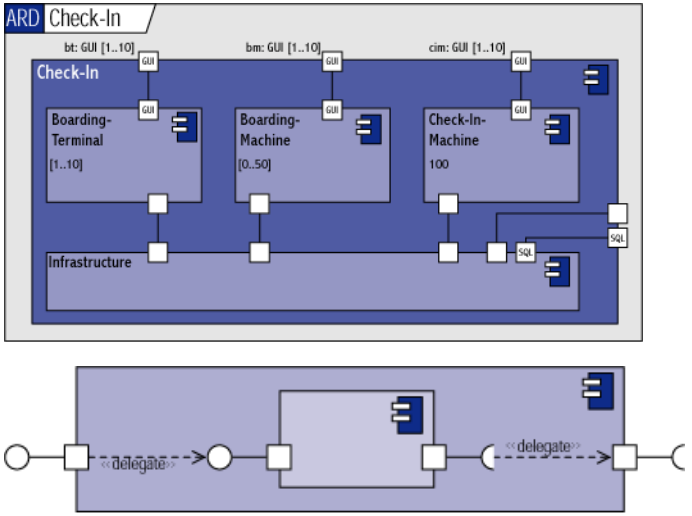


4 - Arquitectura
Un primer vistazo

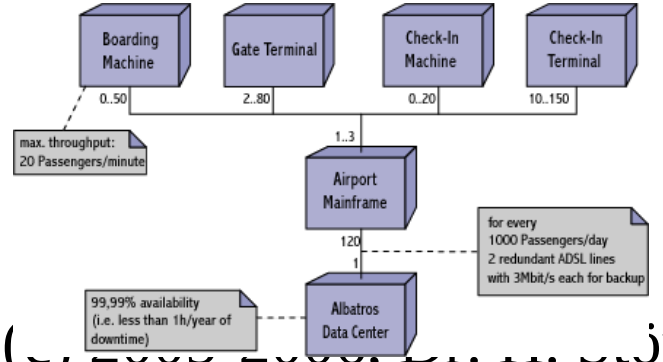
Arquitectura de contexto & dominio



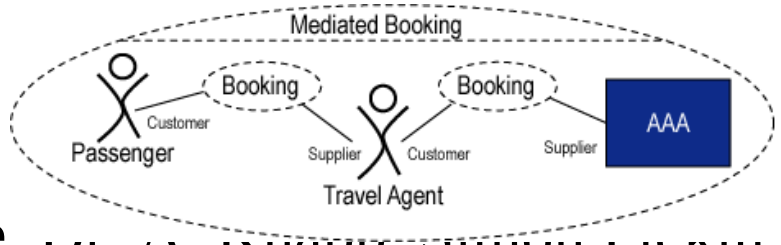
Estructura compuesta diagramas("assembly")



Despliegue

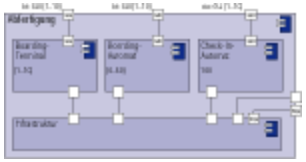


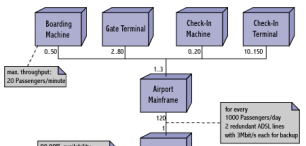
Colaboración



4 - Arquitectura
Escenarios de uso / vistas arquitecturales

- Diagrama de contexto
  - define los límites de un sistema en términos de sus usuarios y sistemas vecinos
  - define nombres/abreviaturas para el sistema y sus vecinos
- Arquitectura de dominio
  - proporciona una visión alto nivel de los componentes del dominio
  - define nombres/abreviaturas para los subsistemas
- Diagrama de estructura compuesta (*system assembly* diagram)
  - define puertos ("interfaces del sistema") con nombres y abreviaturas
  - define conexiones entre interfaces
- Diagrama de estructura compuesta (*class assembly* diagram)
  - como el anterior, en un nivel de granularidad más fino
  - define también interfaces (de lenguaje de programación) para puertos
- Colaboración
  - documenta decisiones de diseño ("patrones") de cualquier nivel de granularidad
- Diagrama de estructura del sistema
  - nodos físicos y conexiones entre ellos

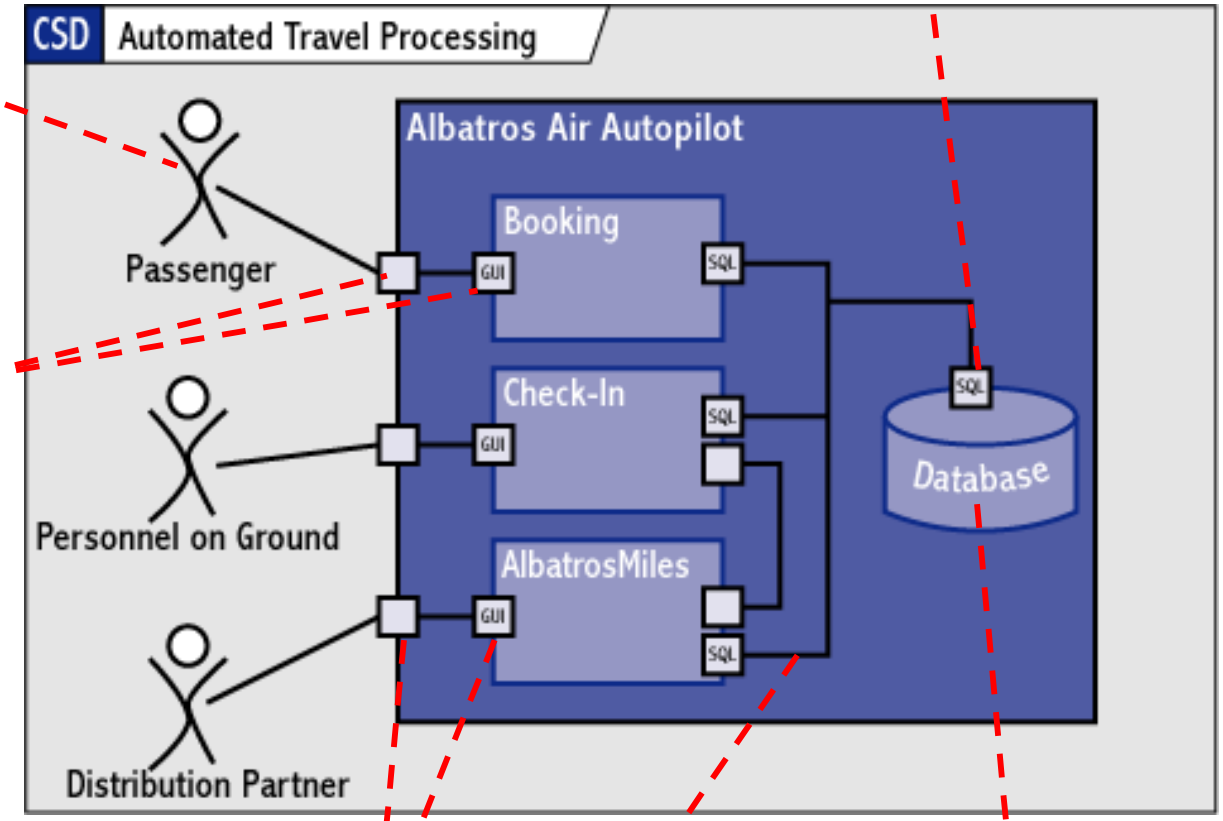




Software de Comunicaciones 2007-2008

4 – Arquitectura
Conceptos principales: estructura compuesta

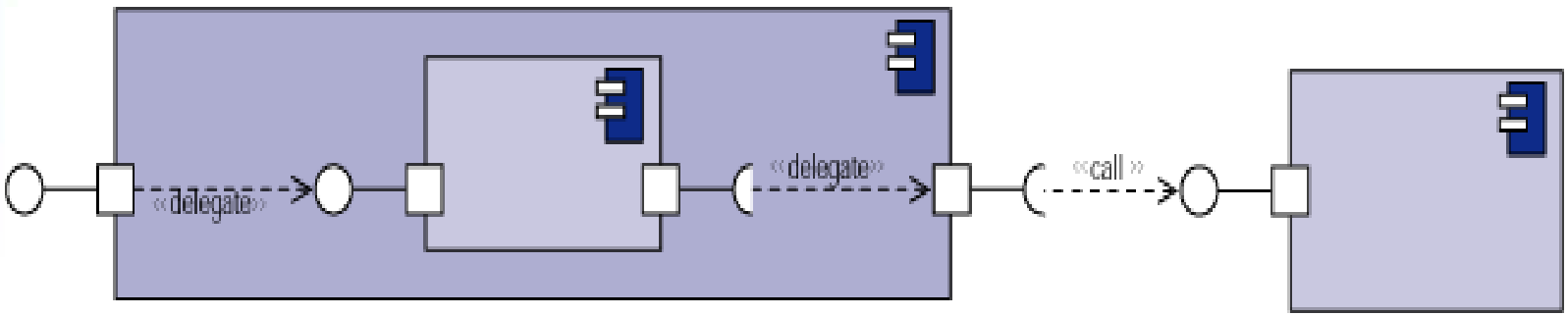
nombre mejor sería: *assembly diagrams*



un sistema tal como una **Clase** o un **Componente**

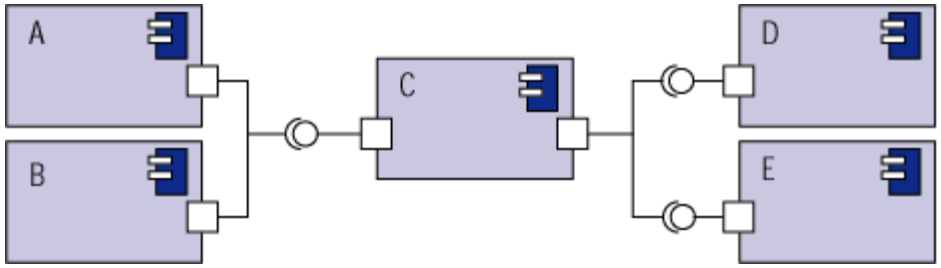
4 – Arquitectura
Uso: conexión de componentes

- Conexión entre puertos equivale a dependencias de tipo delegación o llamada

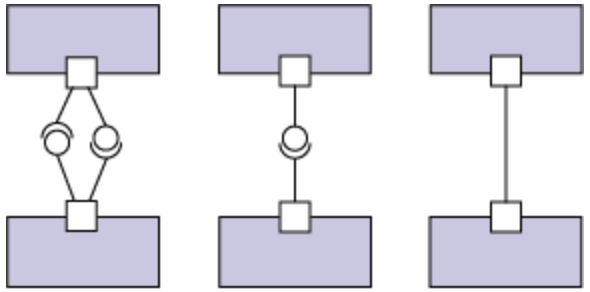


4 - Arquitectura
Uso: conectando componentes

•El uso de la notación conjunta revela detalles sobre el



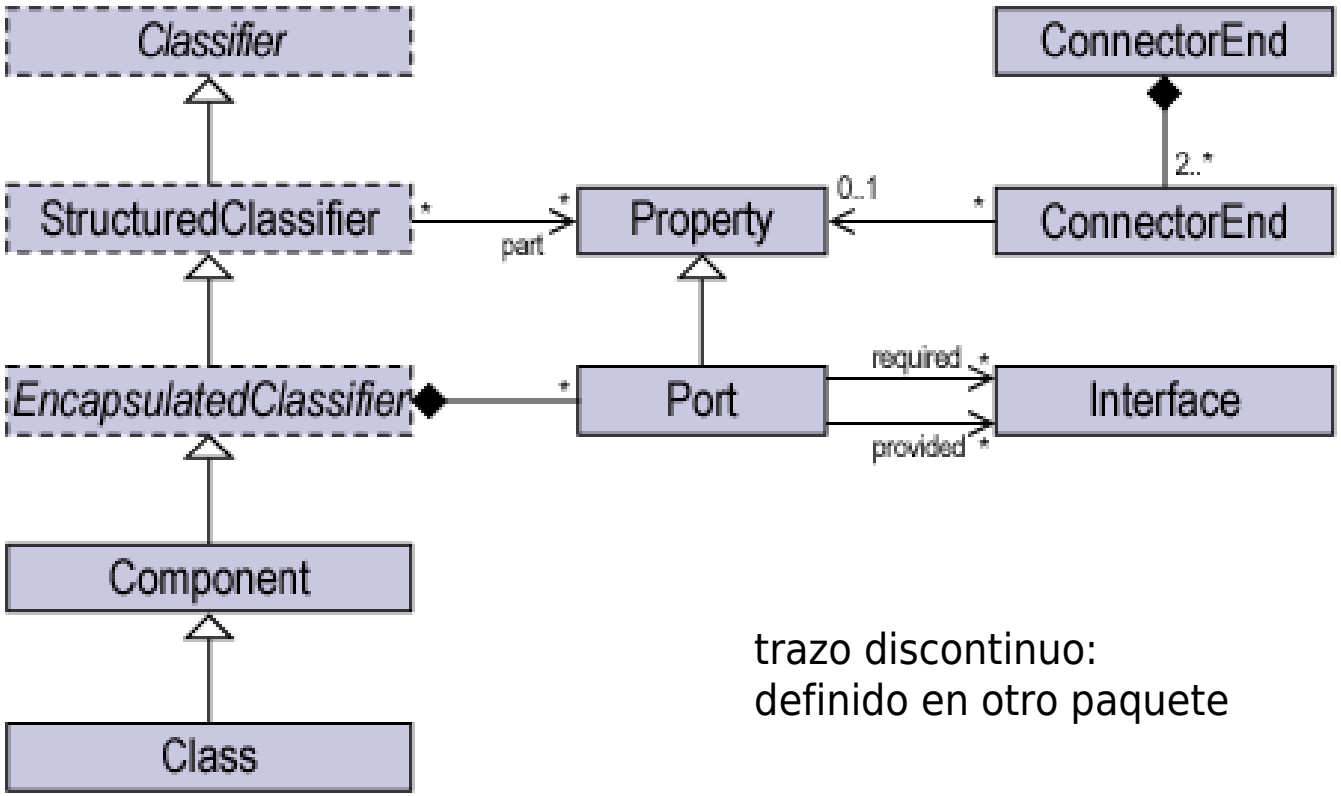
- De izquierda a derecha:
  - dos conectores, uno en cada dirección
  - un conector con dirección
  - y un conector sin dirección.



4 – Arquitectura Componentes

- Los componentes de UML 1.x eran sólo binarios. En UML 2, los componentes se definen de forma mucho más completa.
  - "Un componente representa una parte modular de un sistema que encapsula sus contenidos y cuya manifestación es reemplazable dentro de su entorno.
  - Un componente define su comportamiento en términos de las interfaces propòrcionadas y requeridas. Como tal, un componente sirve como un tipo, cuya conformidad se define mediante estas interfaces requeridas y proporcionadas (que describen su semántica tanto estática como dinámica). Un componente puede por tanto substituirse por otro sólo si los dos tienen tipos conformes. […]
  - Un componente se modela a través de todo el ciclo de desarrollo […]."

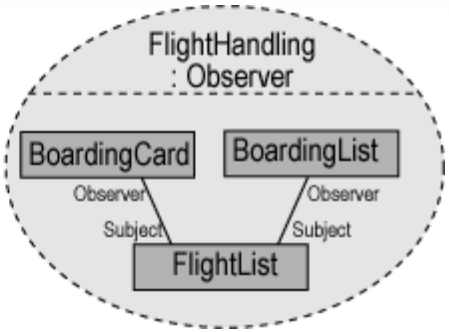
4 – Arquitectura
Metamodelo: Partes y puertos

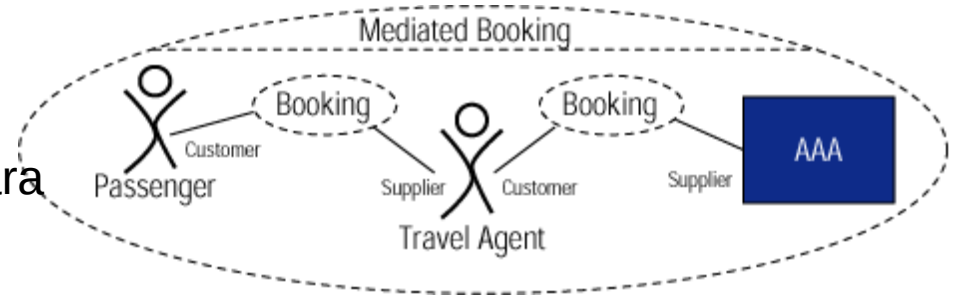


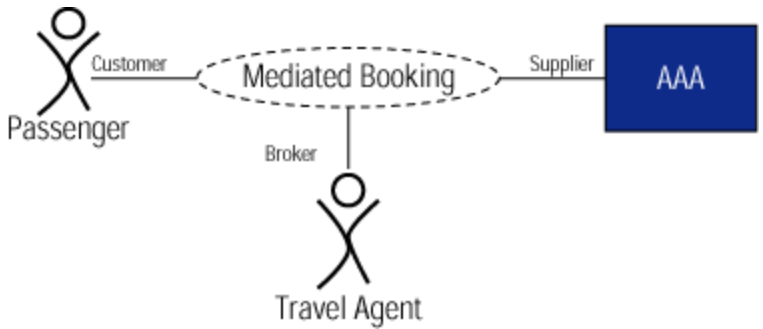
trazo discontinuo:
definido en otro paquete

4 - Architecture
Colaboración

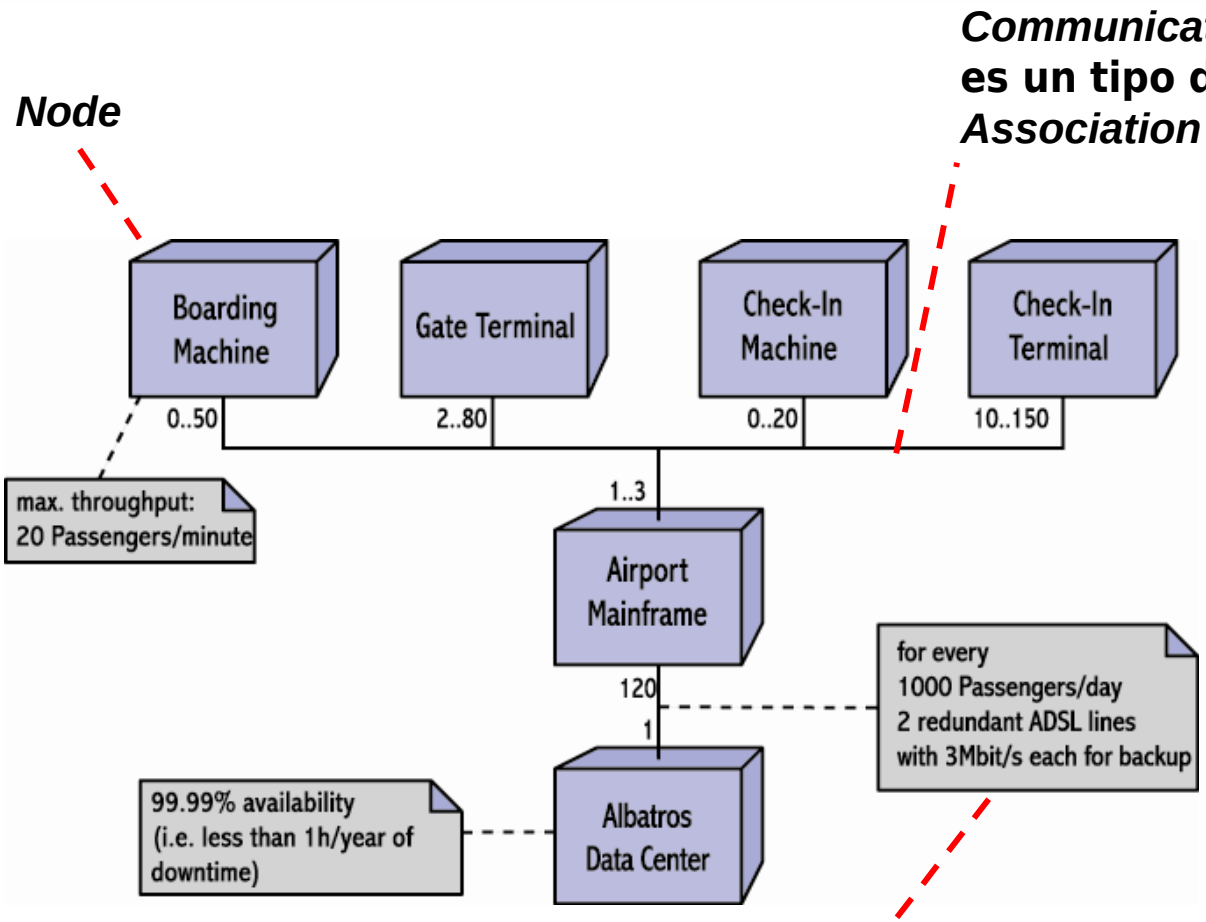
- Definen de forma abstracta un tipo de vinculación sin ser específico.
- Se pretende que es la forma de describir patrones de análisis y diseño.
- Pueden ayudar en etapas tempranas del diseño arquitectónico.
- Podrían también usarse para describir restricciones globales.
- Pueden anidarse y componerse.
- Metodologicamente, son los equivalentes estructurales de los casos de uso.





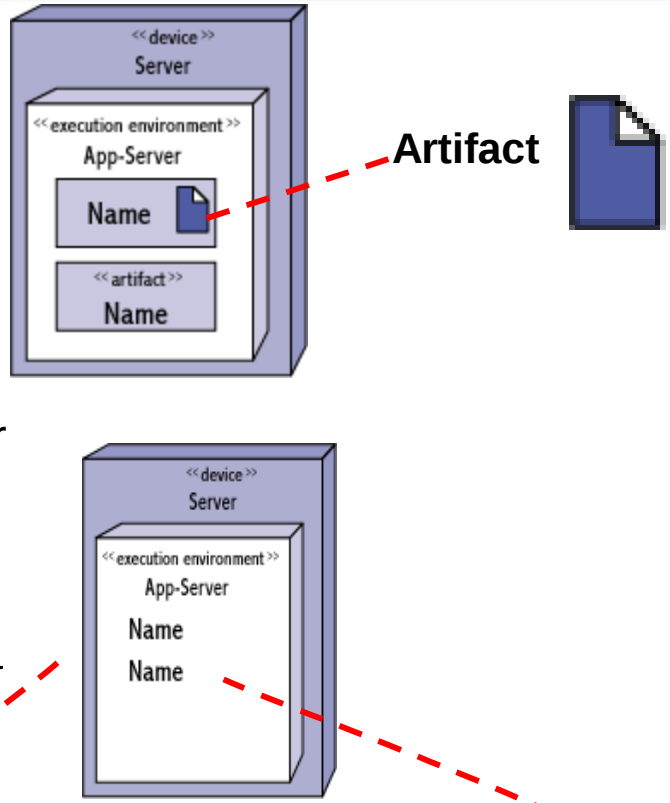


4 – Arquitectura
Estructura del sistema

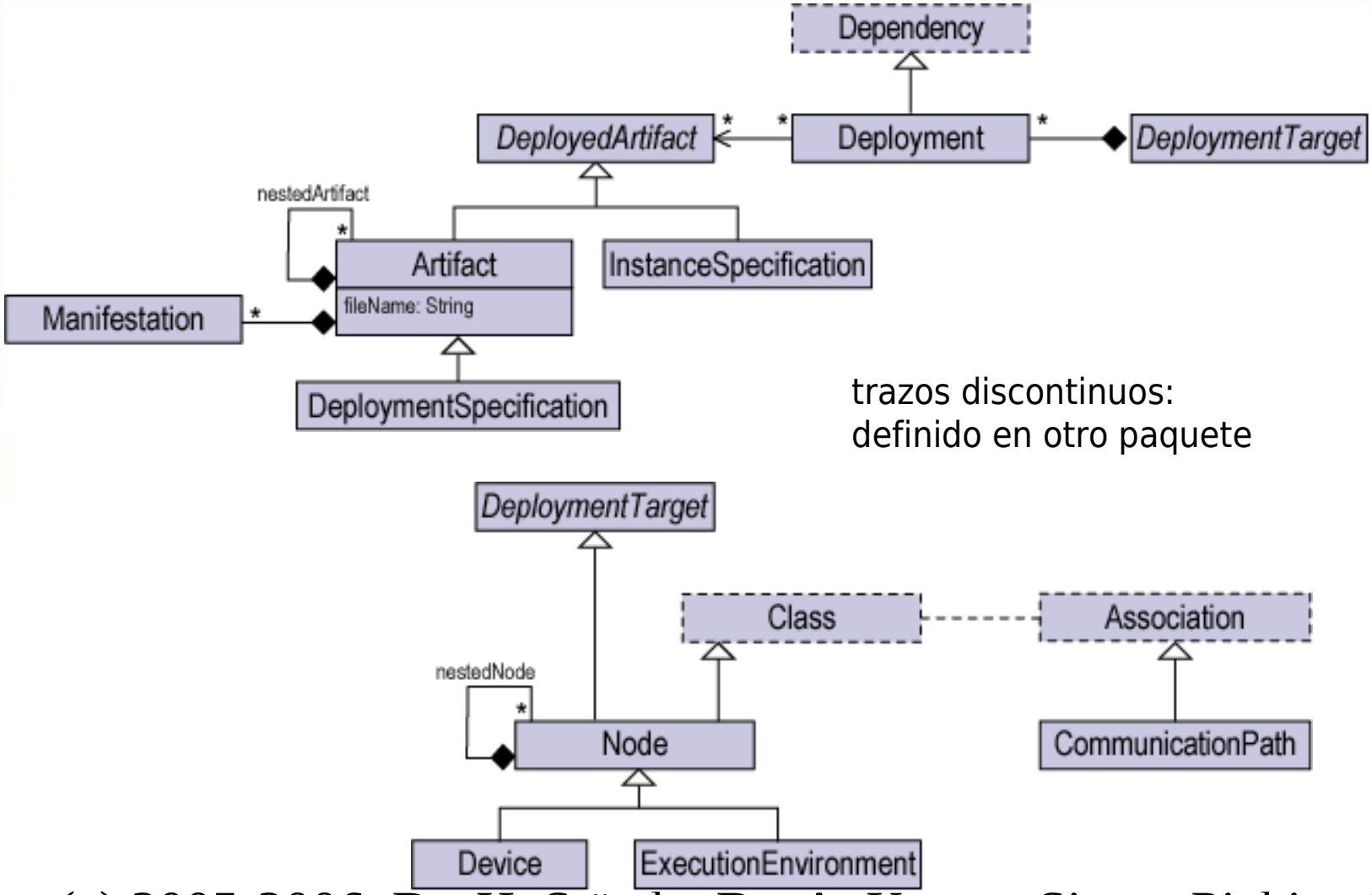


4 – Arquitectura
Despliegue

- Un despliegue es una correspondencia entre artefactos y nodos del sistema.
  - pueden incluir
    - binarios
    - instancias de componente
  - los nodos del sistema pueden incluir
    - hardware (*Dispositivo*)
    - software (*ExecutionEnvironment*)
- Formalmente, un despliegue es una *DeploymentRelationship*.
- Puede representarse situando las entidades desplegadas o sus nombres sobre el objeto de despliegue.



4 – Arquitectura
Metamodelo: Despliegue

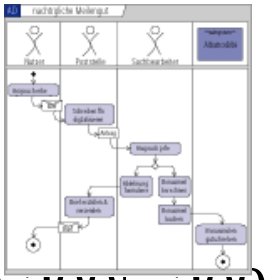


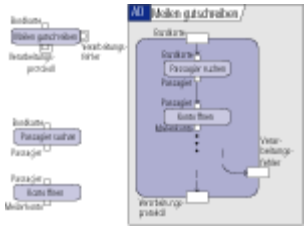
4 - Arquitectura
Conclusiones

- Los conceptos populares del modelado arquitectural (“capsula”/“actor”, “puerto”) por fin han sido incluidos en el UML, aunque el metamodelado no es muy sólido.
- Despliegues, artefactos y conceptos relacionados se han extendido, y son ahora ciudadanos de primera clase.
- Los componentes tienen finalmente una definición decente como unidades a través del ciclo de vida.

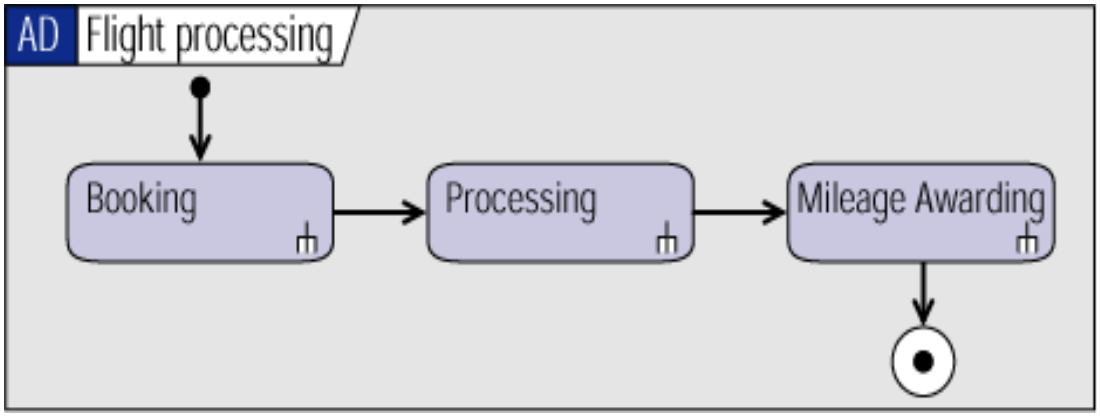
Unified Modeling Language 2

Part 5 - Actividades





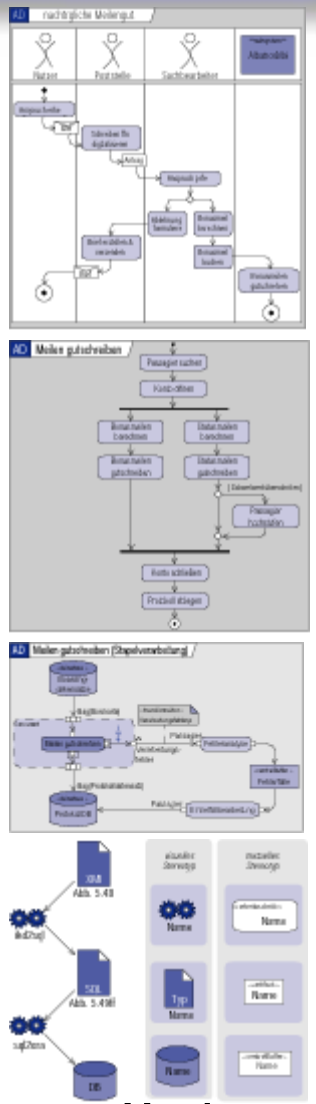
5 - Actividades
Primer vistazo



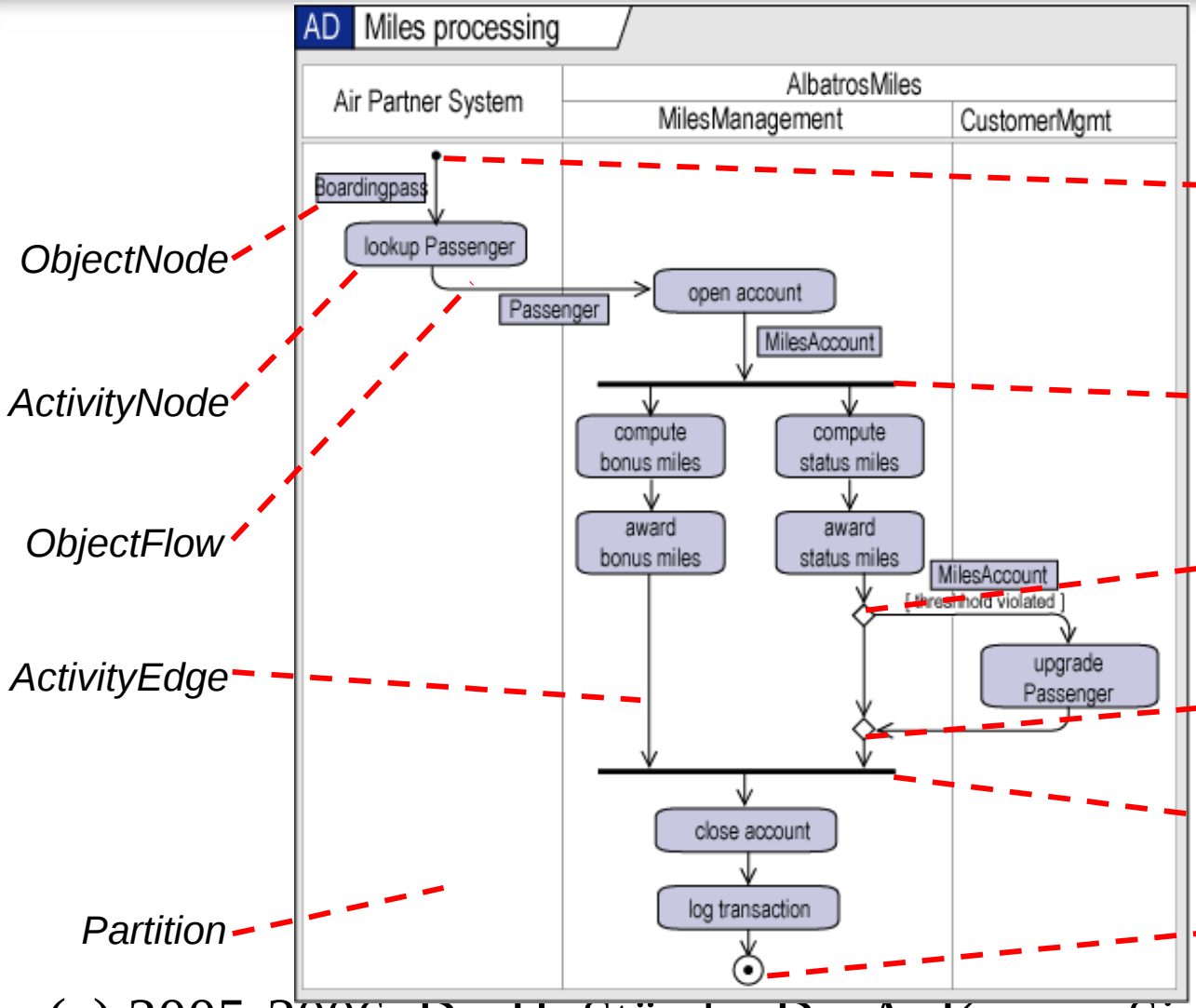
- Los diagrams presentan todo tipo de flujos de control y de flujos de datos.
- Son de alguna manera duales a las máquinas de estado: el enfoque es sobre acciones en vez de estados.
- Se ha guardado y extendido la notación de UML 1.x (aunque cambiando el significado).

5 - Actividades
Escenarios de uso

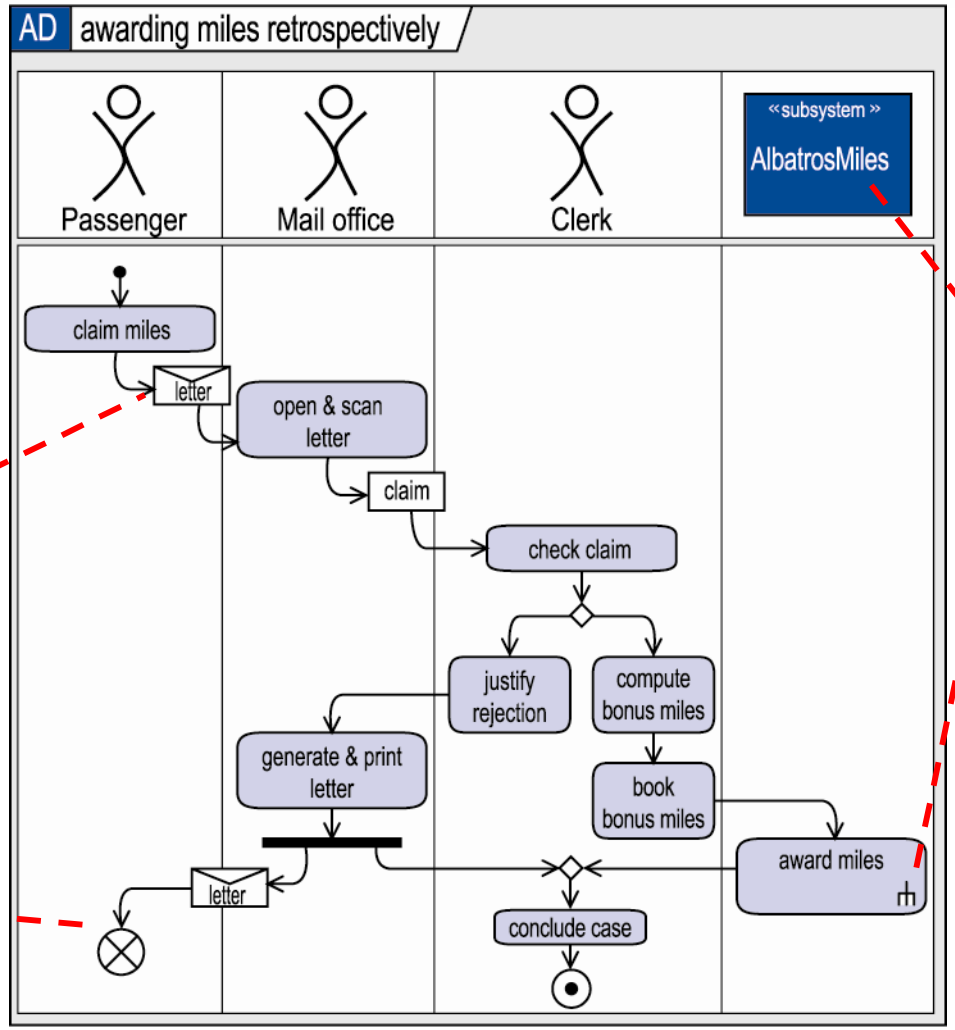
- Los diagramas de actividad se pueden aplicar con múltiples fines a través de todo el ciclo de vida.
- Análisis
  - diseñar o documentar procesos en el dominio de aplicación (*procesos de negocio*)
- Diseño
  - diseñar o documentar procesos como composiciones de elementos pre-existentes tales como tareas manuales o tareas a automatizar.
- Implementación
  - documentar programas existentes (es decir, funciones, servicios, …)
  - diseñar procesos algoritmicos con la intención de convertirlos en código en un lenguaje de implementación



5 - Actividades
Conceptos principales



5 - Actividades
Conceptos principales



ObjectNode estereotipado

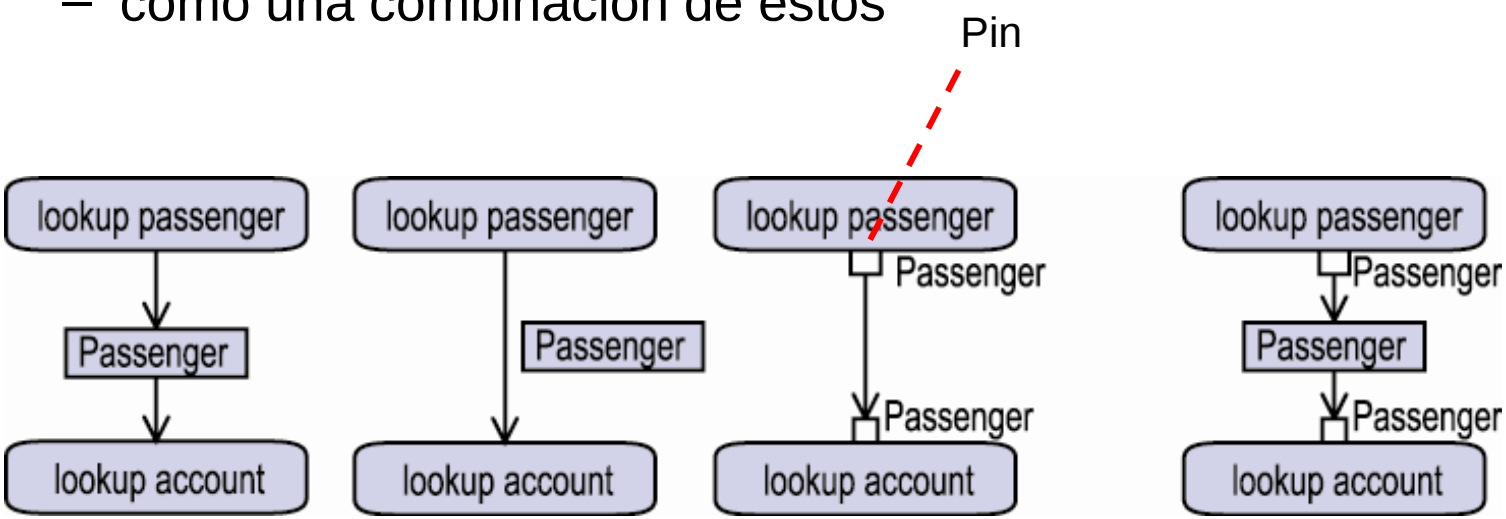
FlowFinal

Las partitciones pueden representarse explícitamente

Activity refinado

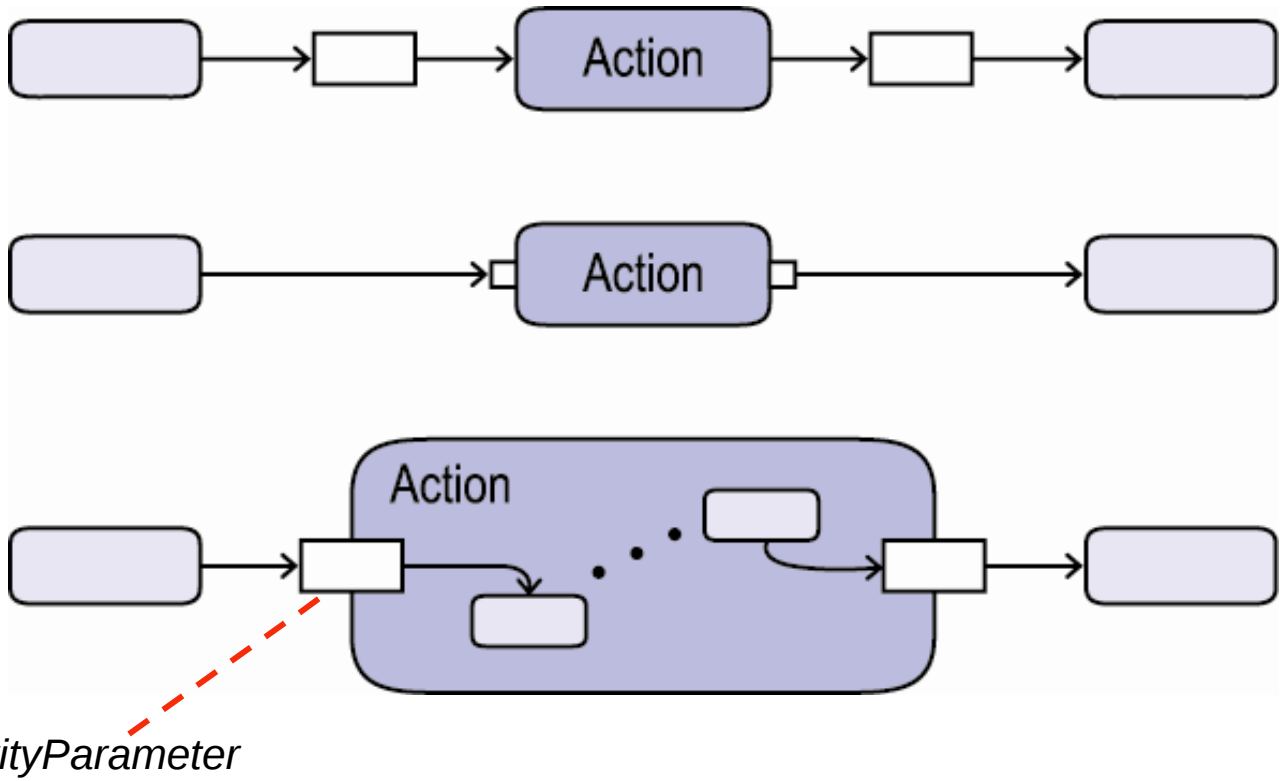
5 - Actividades
Pins

- Los flujos de datos pueden representarse
  - explícitamente,
  - con nodos de flujo de datos unidos a un flujo de control,
  - con *Pins* en las Actividades, o
  - como una combinación de éstos



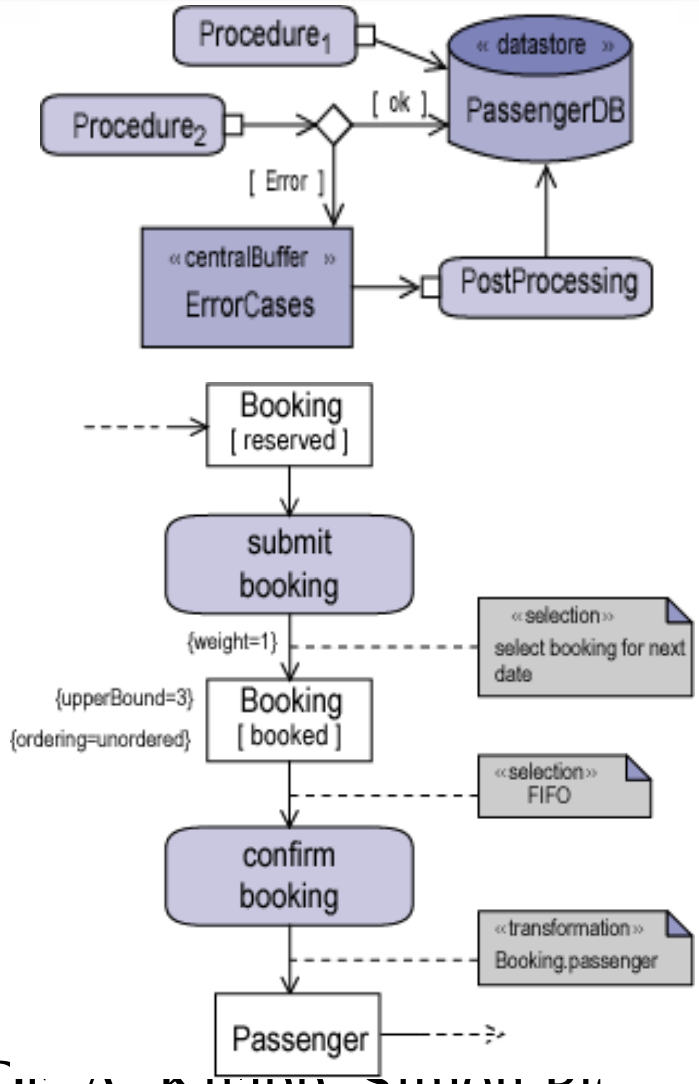
5 - Actividades
Parámetros de actividades

- *Los pins* actúan como parámetros para actividades.



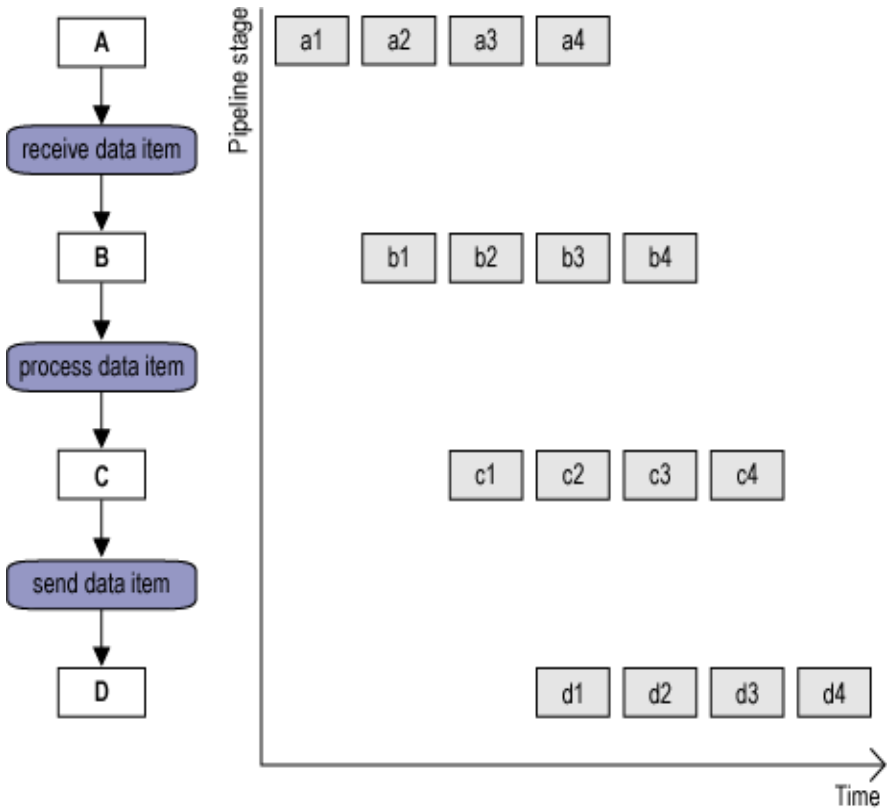
5 - Actividades
Detalles de flujos de datos

- Un flujo de datos define el transporte de datos entre buffers por **a**ctividades.
- Los buffers pueden tener capacidades y estar ordenados.
- A parte del transporte como tal, un flujo de datos puede definir también
  - la selección de un dato particular, y
  - la transformación de datos.
- Muchas veces es útil denotar el estado de un dato en un buffer.



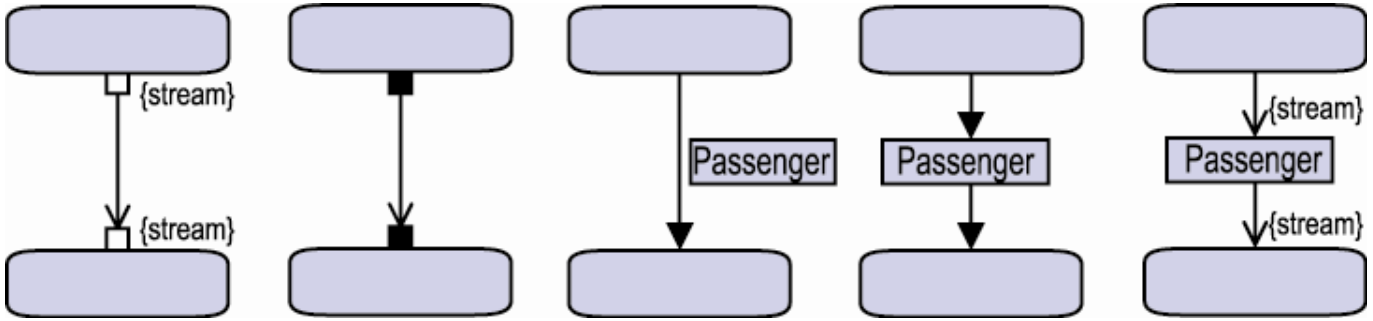
5 – Actividades Streaming

- *Streaming* significa que los datos se procesan en flujo continuo.
- El comportamiento de tipo *stream* no pudo expresarse en UML 1.x.
- *Streaming* se expresa por
  - *Pins* sólidos negros
  - *una anotación explícita en los pins.*
  - cabezas de flecha negras, o
  - con modo *stream* en una región de expansión.

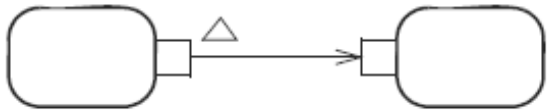


5 - Actividades
Pins de streaming y de excepción

- Notaciones equivalentes para pins de *streaming*.



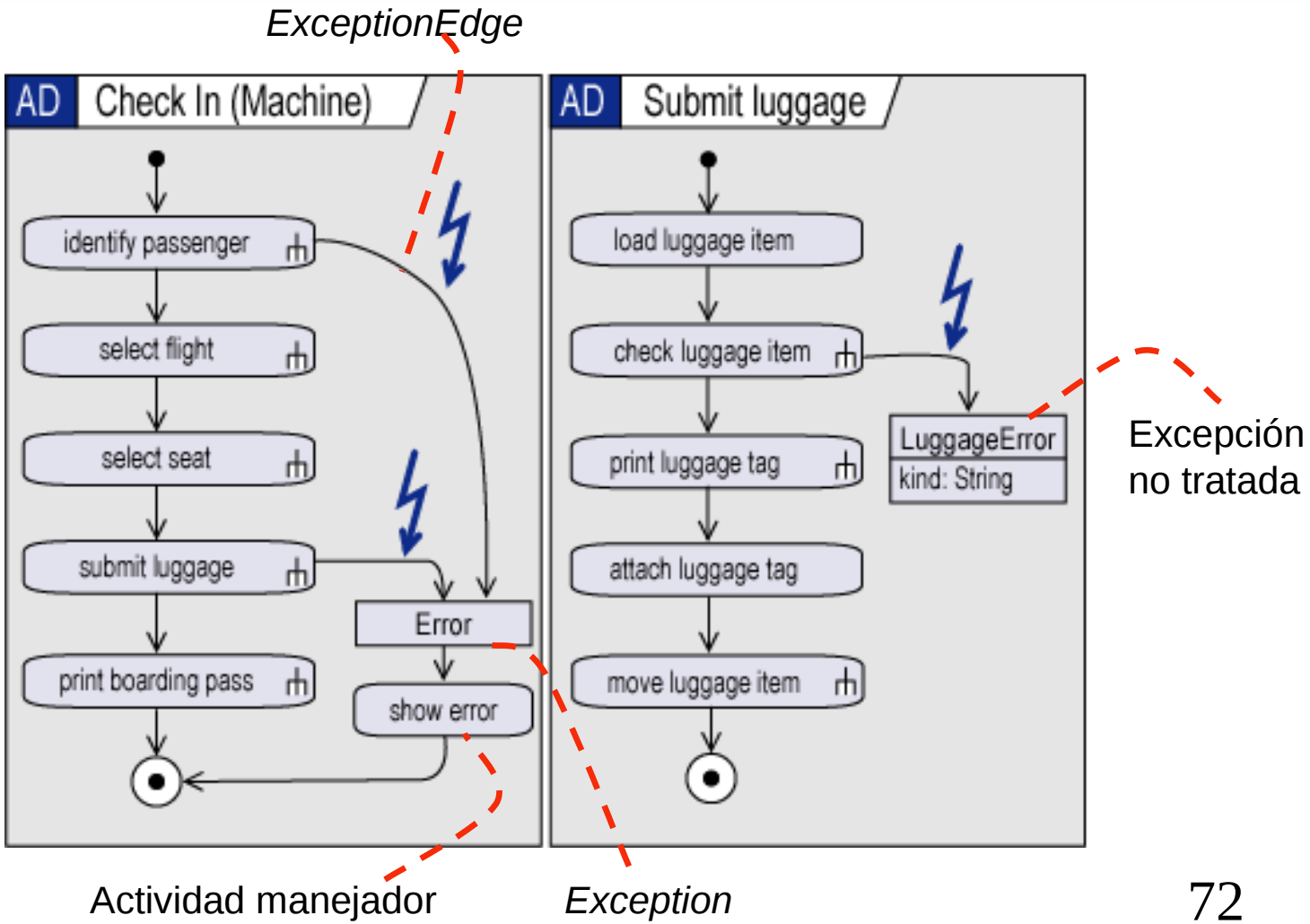
- Notaciones equivalentes para excepciones.



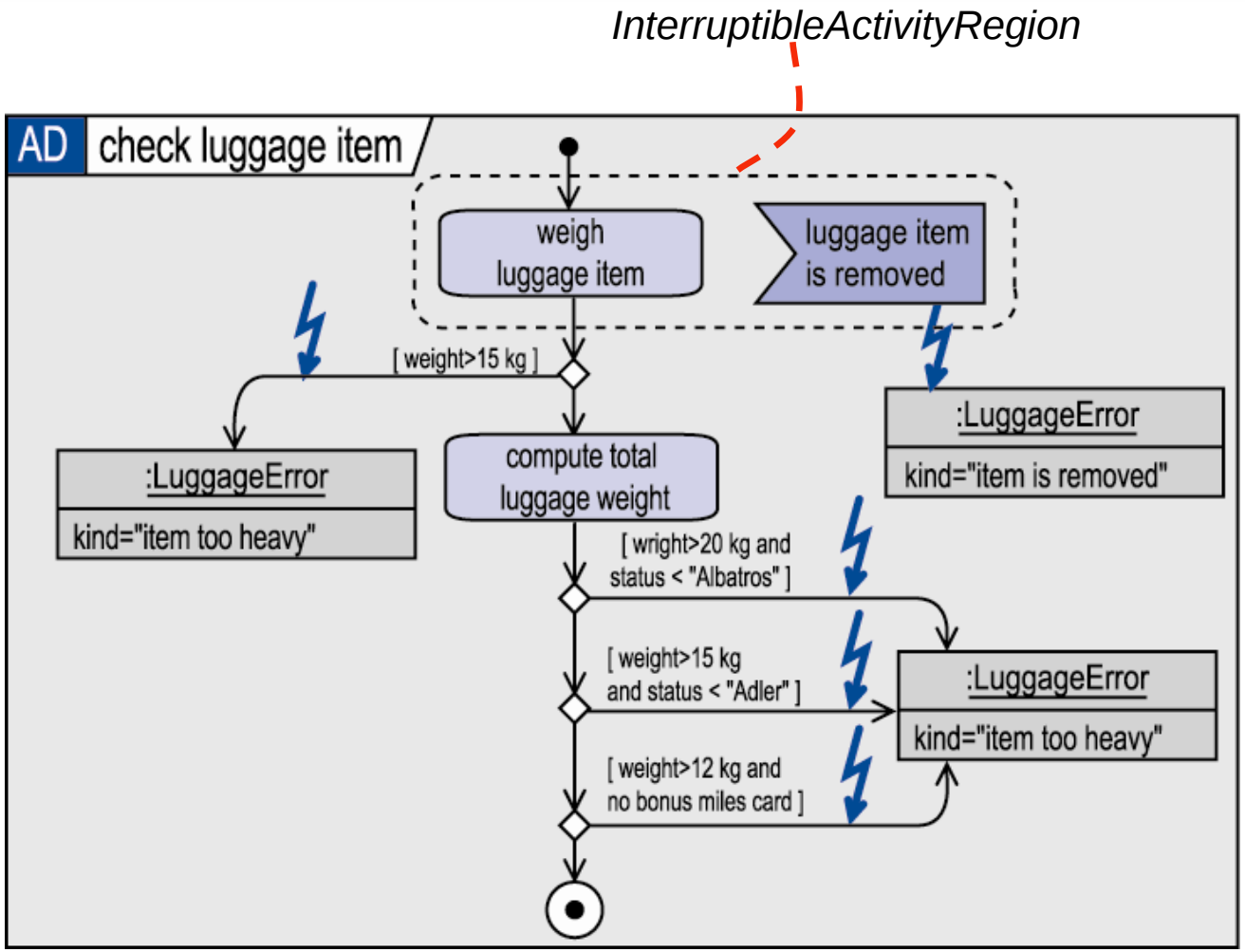
Output pin, pin style, exception

Input and output pin, standalone style, exception

5 - Actividades
Excepciones

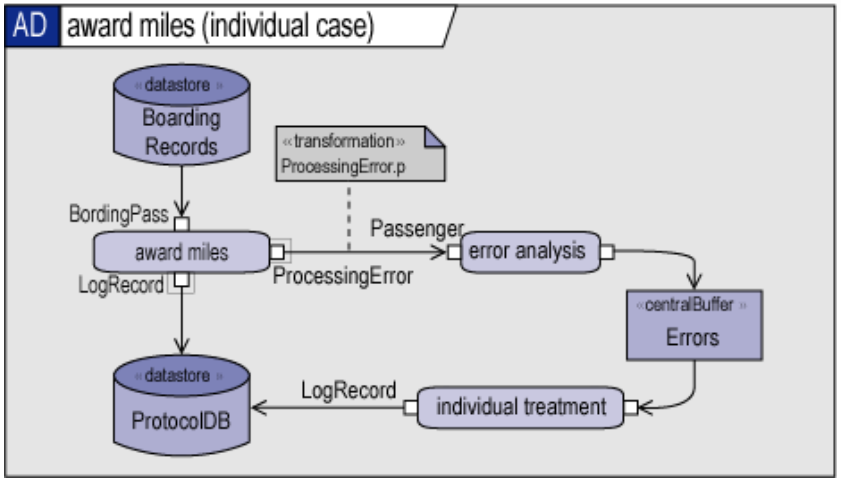


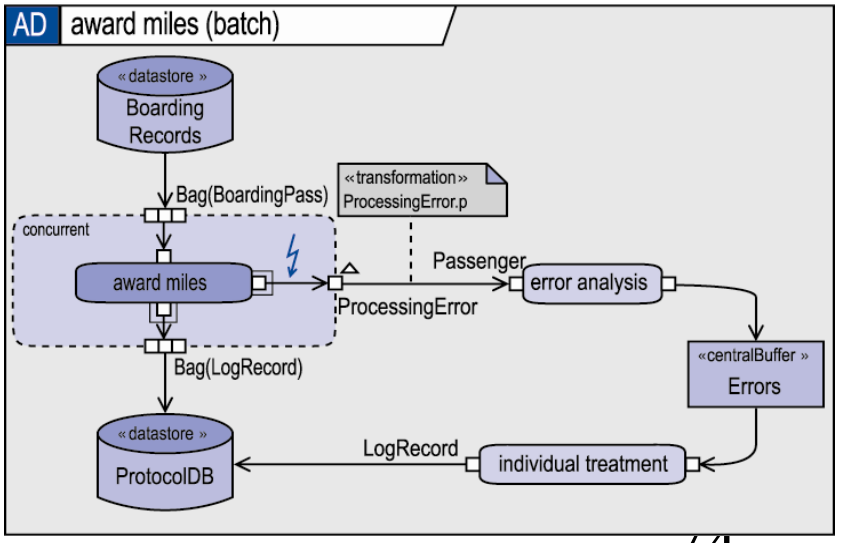
5 - Actividades
Lanzar excepciones



5 - Actividades
Regiones de expansión para el procesamiento de datos por lotes

- Las actividades con frecuencia se usan para especificar procesamiento de collecciones de datos (*lotes*) y no de unidades individuales.
- UML ofrece *ExpansionRegions* para soportar esta posibilidad.
- Una región de expansión declara la aplicabilidad de una actividad a todo una colección de unidades de datos.

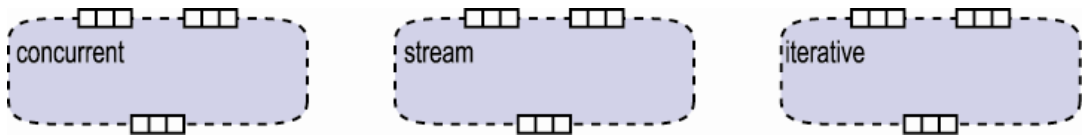


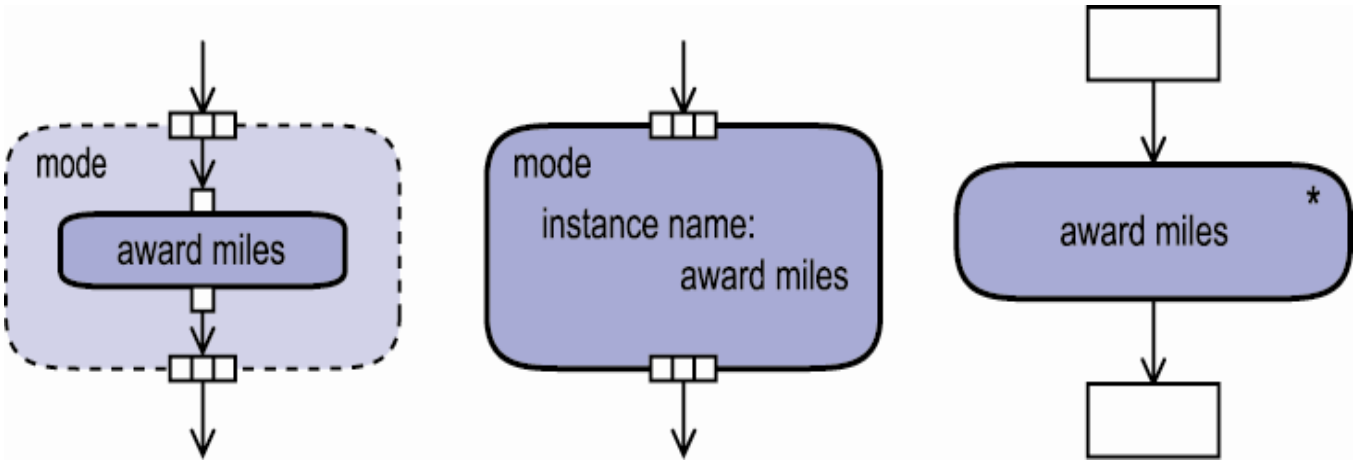


5 - Actividades
Regiones de expansión

Una región de expansión puede procesarse de uno de los siguientes modos

- iterativo,
- concurrente,
- stream.





5 - Actividades
Nodos estructurados

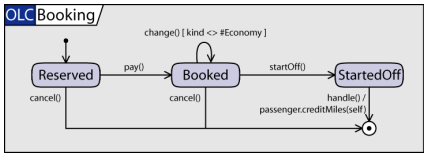
- Un nodo estructurado es aquel que puede expansionarse en nodos subordinados
- Existen nodos estructurados para representar
  - secuencias,
  - bucles,
  - condicionales.
- Sintaxis inexistente/insuficiente (y no digamos semántica) definida por el estándar.

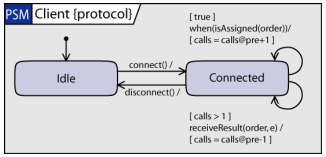
5 - Actividades
Conclusiones

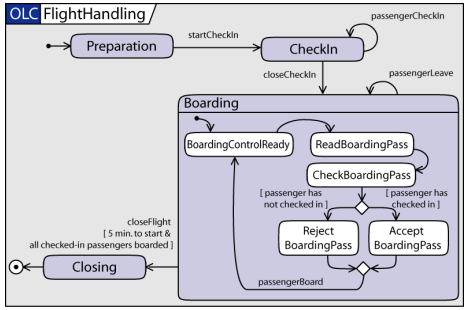
- Presentan flujos de control y de datos para modelos del nivel de análisis, diseño e implementación.
- Ya no son tipos especiales de *StateMachine* como en UML 1.x.
- La semántica está inspirada en las Redes de Petri, aunque actualmente no está del todo clara.
- Introducen muchos conceptos y notaciones nuevos, entre otros
  - gestión de excepciones
  - *streaming* de datos
  - gestión de datos de colecciones
  - nodos estructurados
  - notación de *pins* para flujos de datos.
- En definitiva: los diagramas de actividad son ahora ***el*** lenguaje de descripción de algoritmos – y no sólo dentro del UML

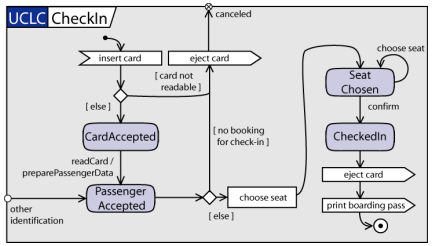
Unified Modeling Language 2

Part 6 – Máquinas de estados



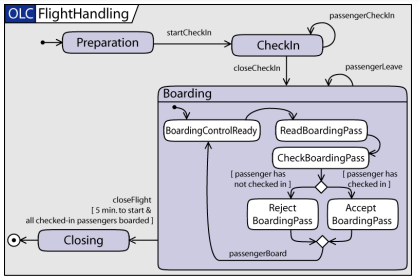


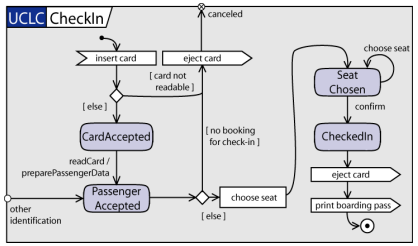


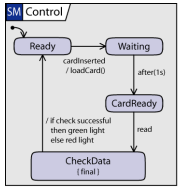


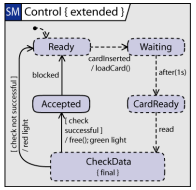
6 – Máquinas de estados
Escenarios de uso

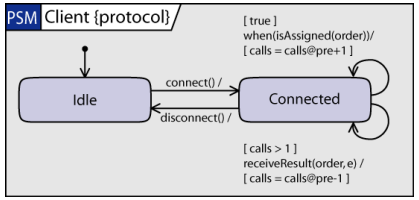
- Ciclo de vida de objeto
  - comportamiento de objetos de acuerdo a reglas de negocio
  - en particular, para clases activas
- Ciclo de vida de caso de uso
  - integración de escenarios de casos de uso
  - alternativa: diagramas de actividad
- Autómata de control
  - sistemas embebidos
- Especificación de protocolos
  - interfaces de comunicación





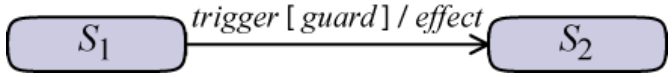


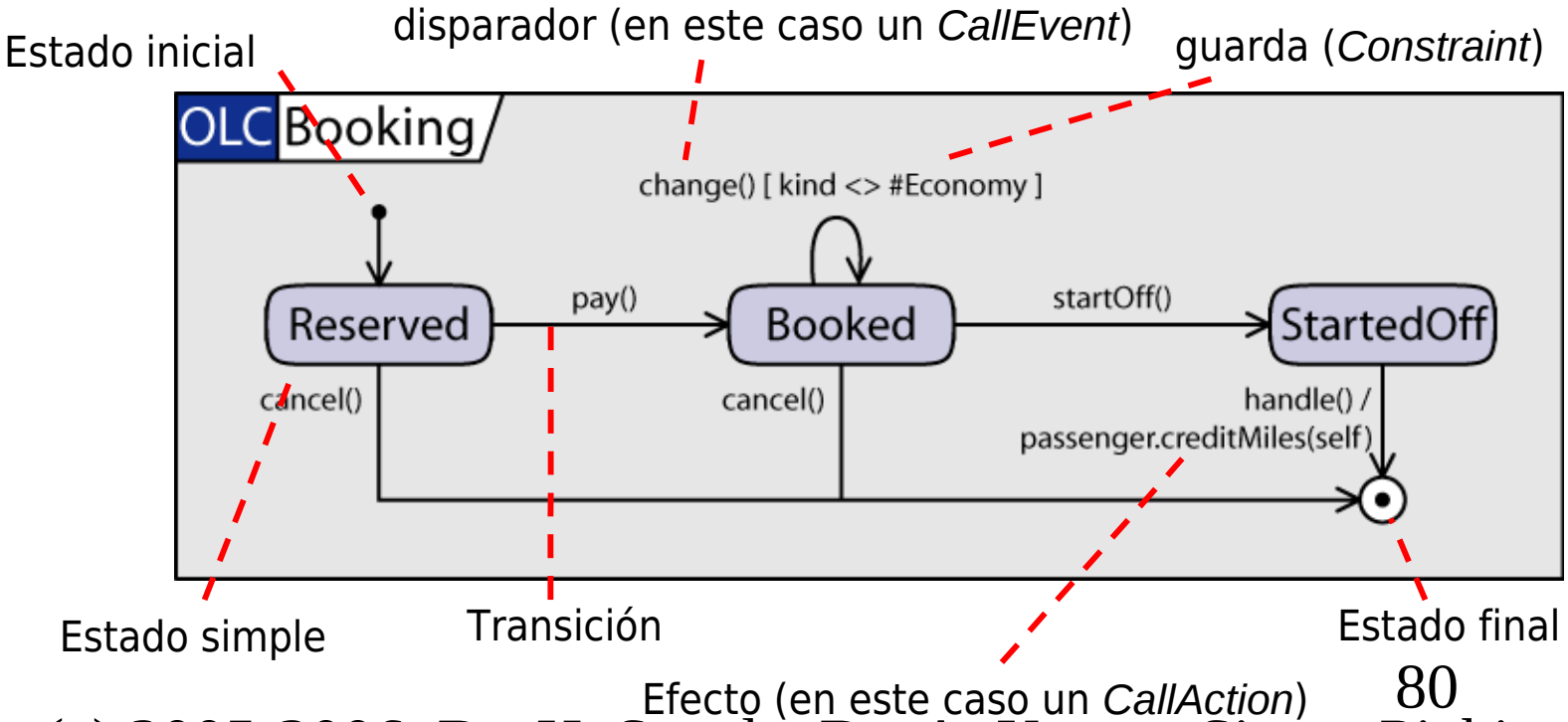




6 – Máquinas de estados
Estados y transiciones

- Las máquinas de estado modelan comportamiento
  - mediante estados interconectados …
  - con transiciones disparadas …
  - por ocurrencias de eventos.





6 – Máquinas de estados
Relación con diagramas de clase

- Las máquinas de estado se definen en el contexto de un *BehavioredClassifier*.

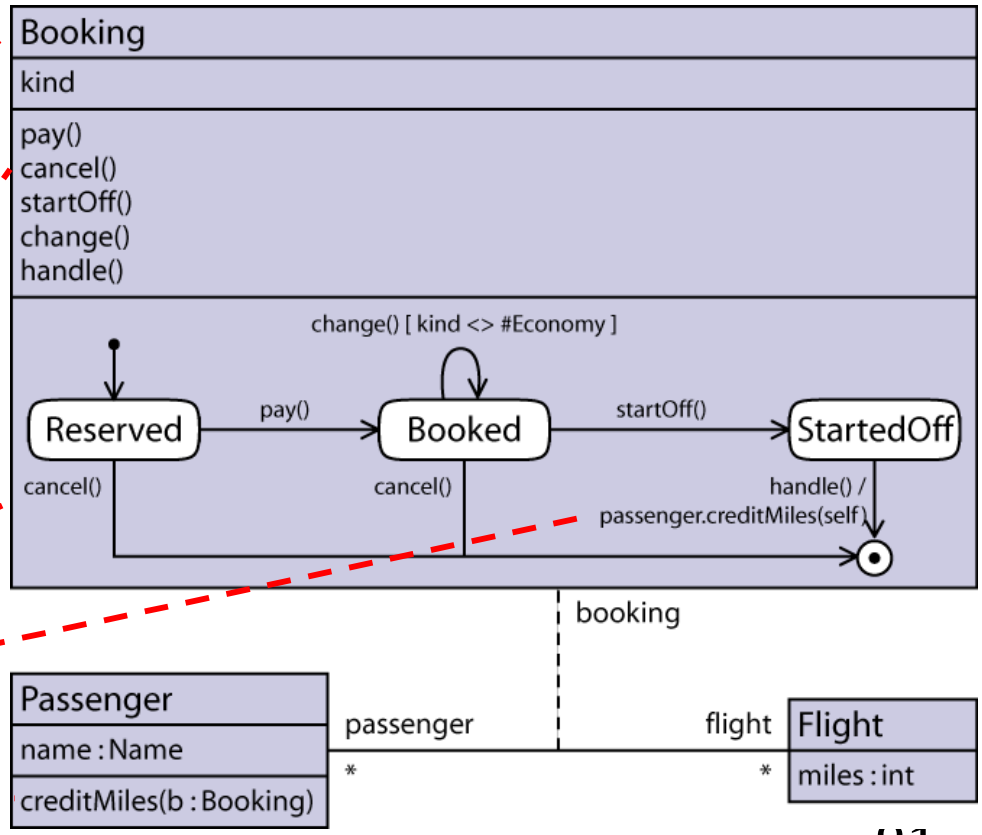
El contexto define qué
- eventos pueden ocurrir
- *features* están disponibles

Operación

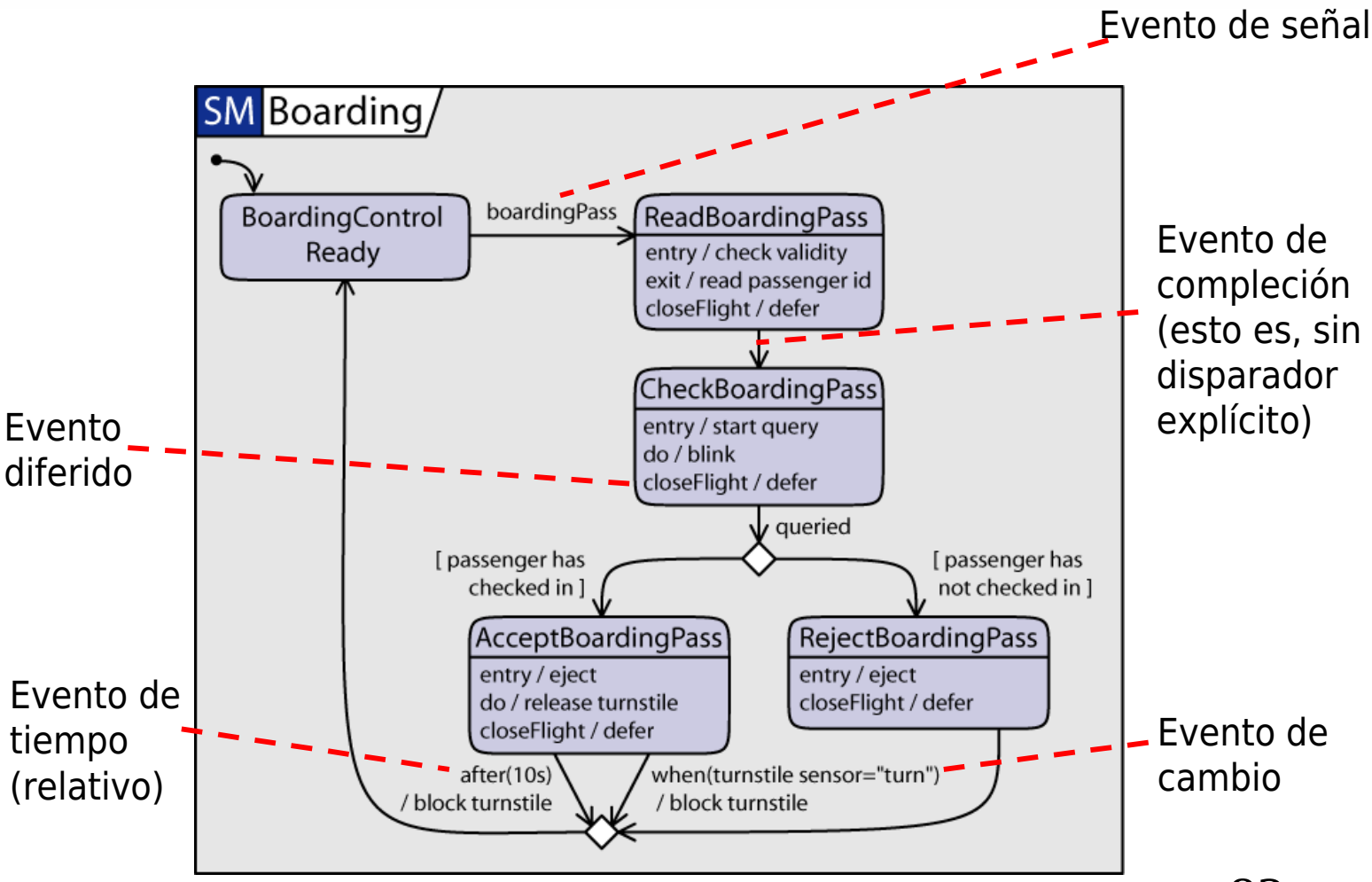
CallEvent provocado por la invocación de la operación

CallAction

Operación llamada por la *CallAction*



6 – Máquinas de estados
Disparadores y eventos (1)

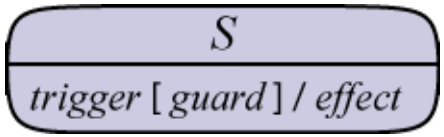


6 – Máquinas de estados
Disparadores y eventos (2)

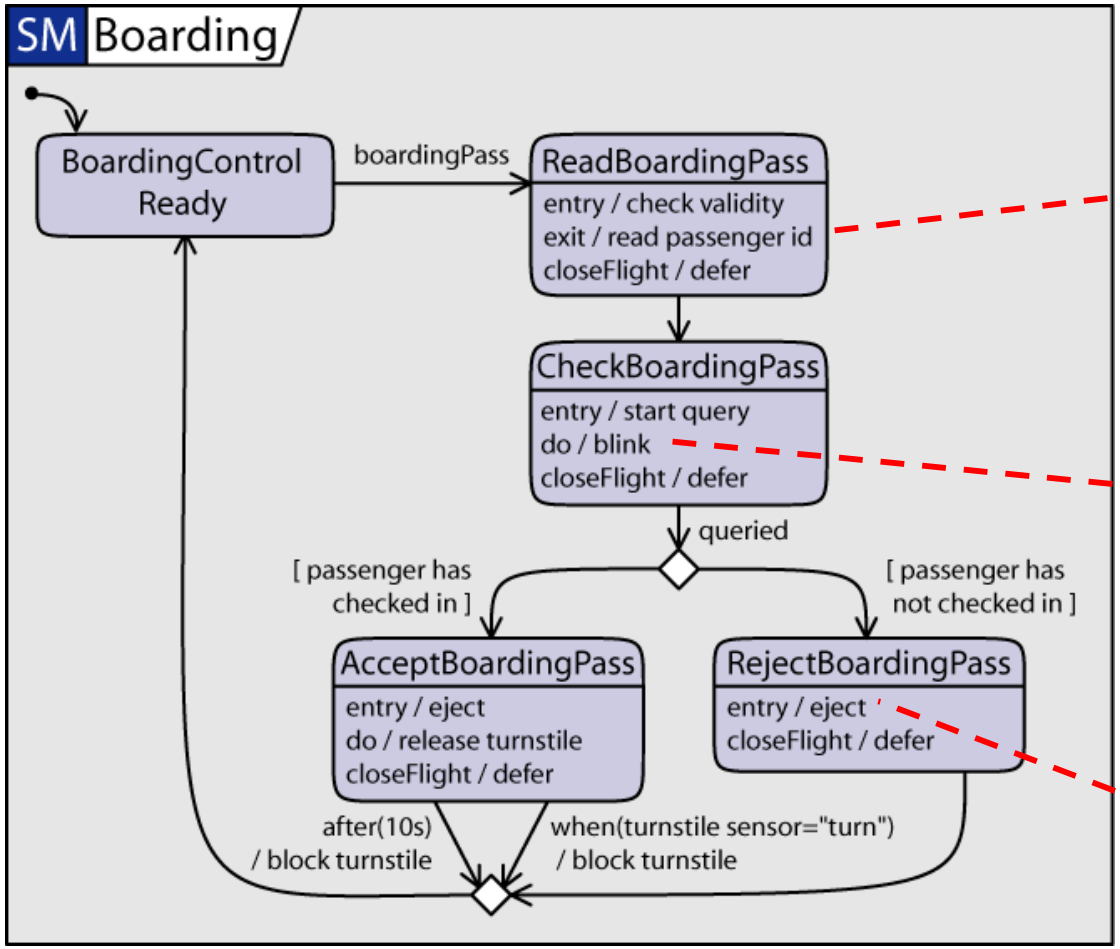
- *Call event* (evento de llamada)
  - recepción de una llamada a una operación síncrona / asíncrona
  - se dispara tras ejecutarse el comportamiento de la operación
- *Signal event* (evento de señal)
  - recepción de una instancia de señal asíncrona
  - reacción declarada mediante una *Reception* para la *Signal*
- *Time event* (evento de tiempo)
  - referencia absoluta a un punto del tiempo (*at t*)
  - referencia relativa a la activación del disparo (*after t*)
    - significado presumiblemente relativo a la entrada al estado
- *Change event* (evento de cambio)
  - cada vez que la condición temporal se hace cierta
    - puede producirse en cierto punto tras hacerse cierta la condición
    - podría revocarse si la condición se tornase falsa

6 – Máquinas de estados
Disparadores y eventos (3)

- *Completion event* (Evento de compleción)
  - producido cuando terminan todas las actividades internas de un estado
    - *do* actividad, subregión
    - sin elemento del metamodelo asociado
  - dispatched antes del resto de eventos en el pool de eventos
- *Diferred event* (evento diferido)
  - eventos que no pueden gestionarse en un estado pero debieran guardarse en el pool de eventos
    - reconsiderados tras el cambio de estado
    - no existe política de deferencia predefinida
- Transición interna
  - … se ejecutan sin abandonar/ entrar en el estado que las contiene
    - normalmente, durante la ejecución de transiciones se abandonan/entra en estados



6 – Máquinas de estado
Comportamientos

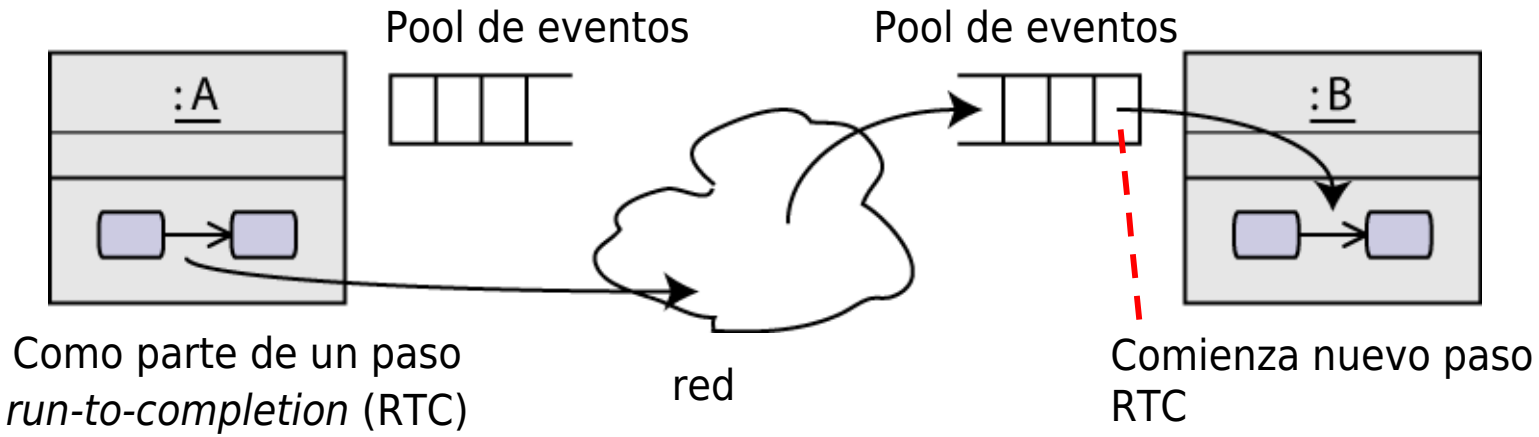


exit Behavior
(al salir
del estado)

do Behavior
(concurrentemente,
mientras la
estancia
en el estado;
puede
interrumpirse)

entry Behavior
(al entrar
en el estado)

6 – Máquinas de estados
Cómo se comunican



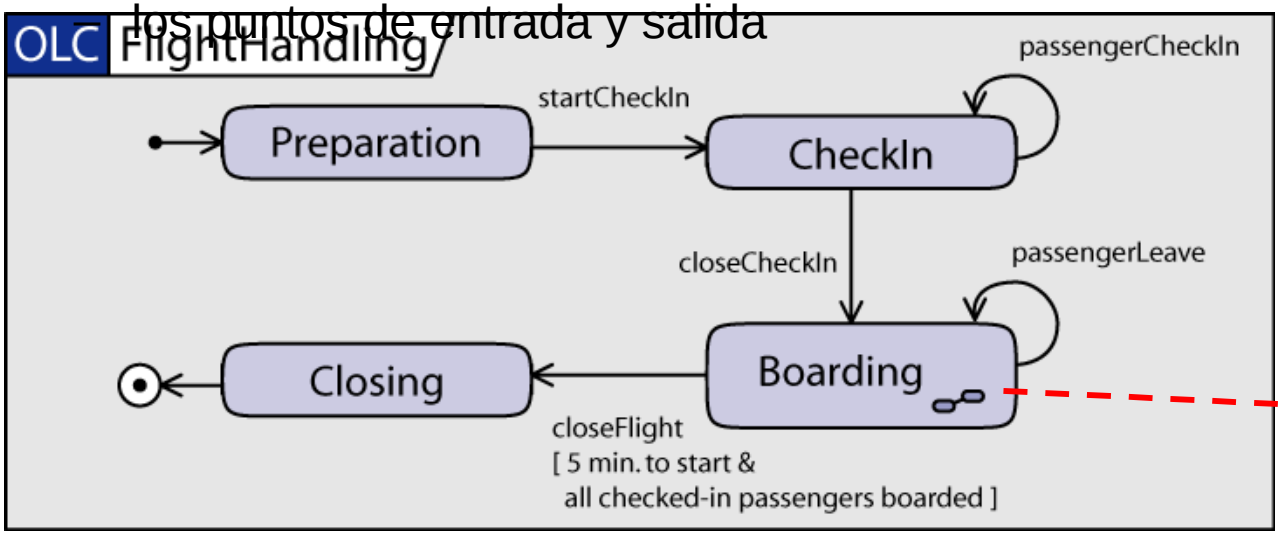
señales: asíncronas (sin espera)
llamadas: asíncronas o síncronas (esperando por RTC del llamado)

No se hacen suposiciones sobre la temporización entre:
la ocurrencia de eventos, *dispatching* de eventos, consumo de eventos.

Pueden descartarse las ocurrencias de eventos para los que no existe disparador (si no se han especificados como diferidos).

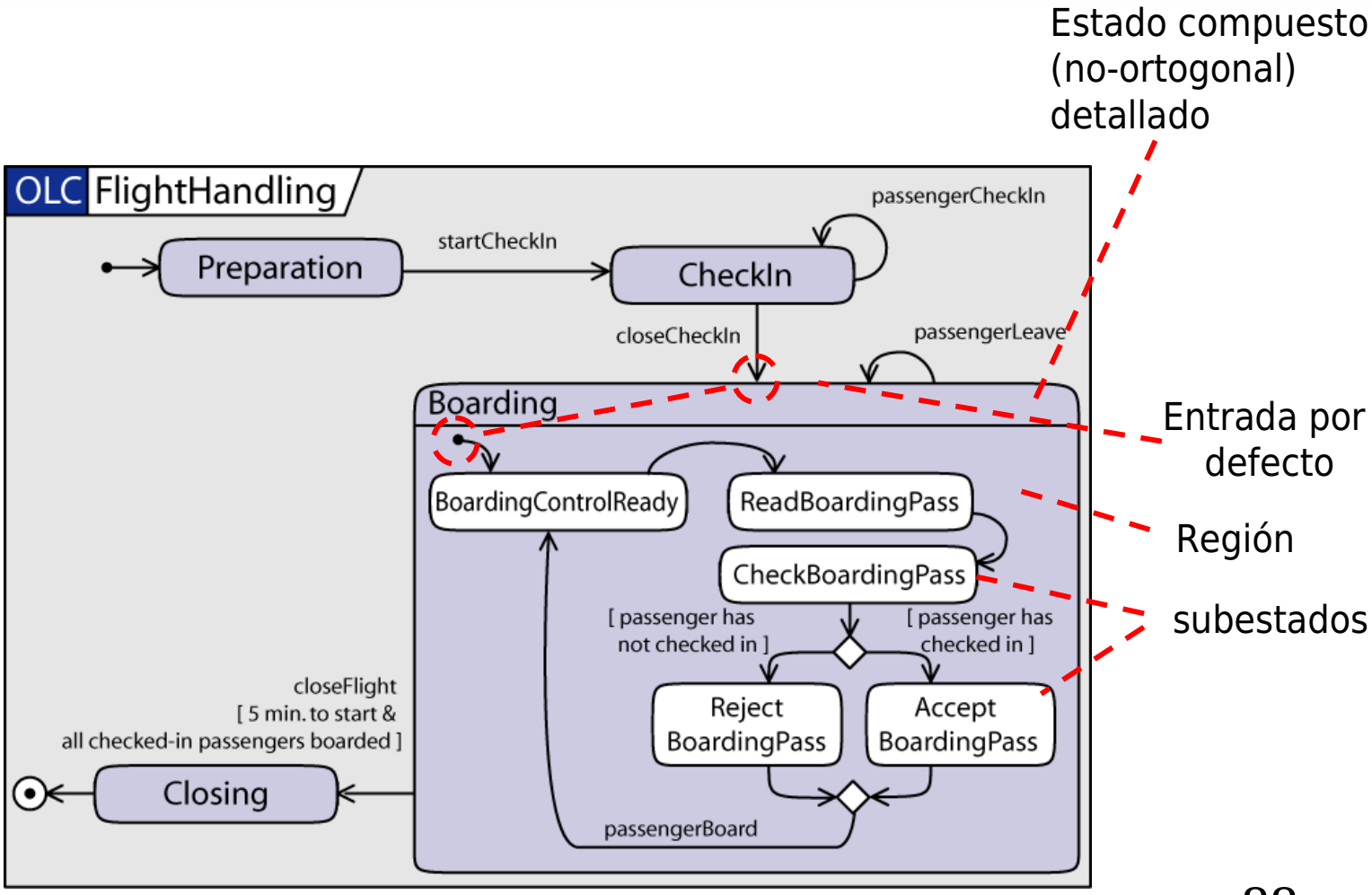
6 – Máquinas de estados
Estados jerárquicos (1)

- La encapsulación del comportamiento de los estados jerárquicos facilita el reuso.
- Sin embargo, en UML 1.x raras veces se aprovecha para reuso.
- UML 2 proporciona conceptos para dar soporte a este uso.
  - los puntos de entrada y salida



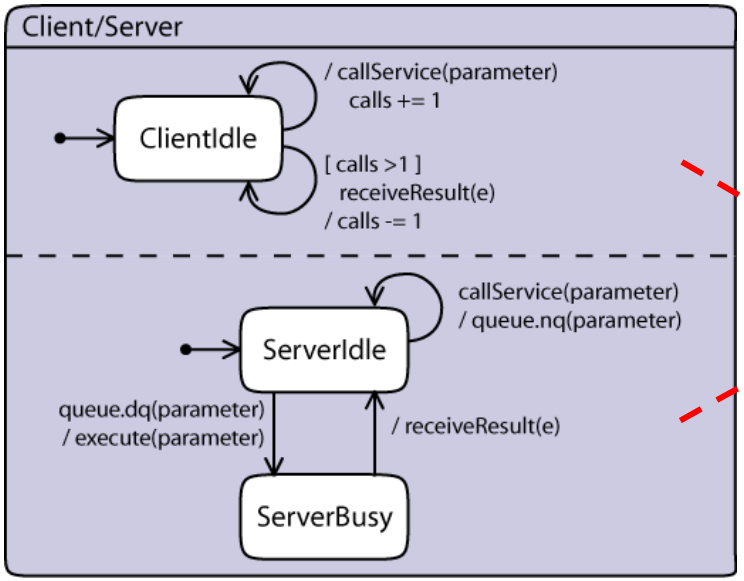
El disparo de transiciones se prioriza de dentro a fuera, es decir, se consideran primero las transiciones más profundas en la jerarquía.

6 – Máquinas de estados
Estados jerárquicos (2)



6 – Máquinas de estado
Regiones ortogonales

- **Estado simple**: no contiene región alguna
- **Estado compuesto**: contiene al menos una región
  - *estado compuesto simple*: exactamente uno
  - *estado compuesto ortogonal*: al menos dos (concurrencia)



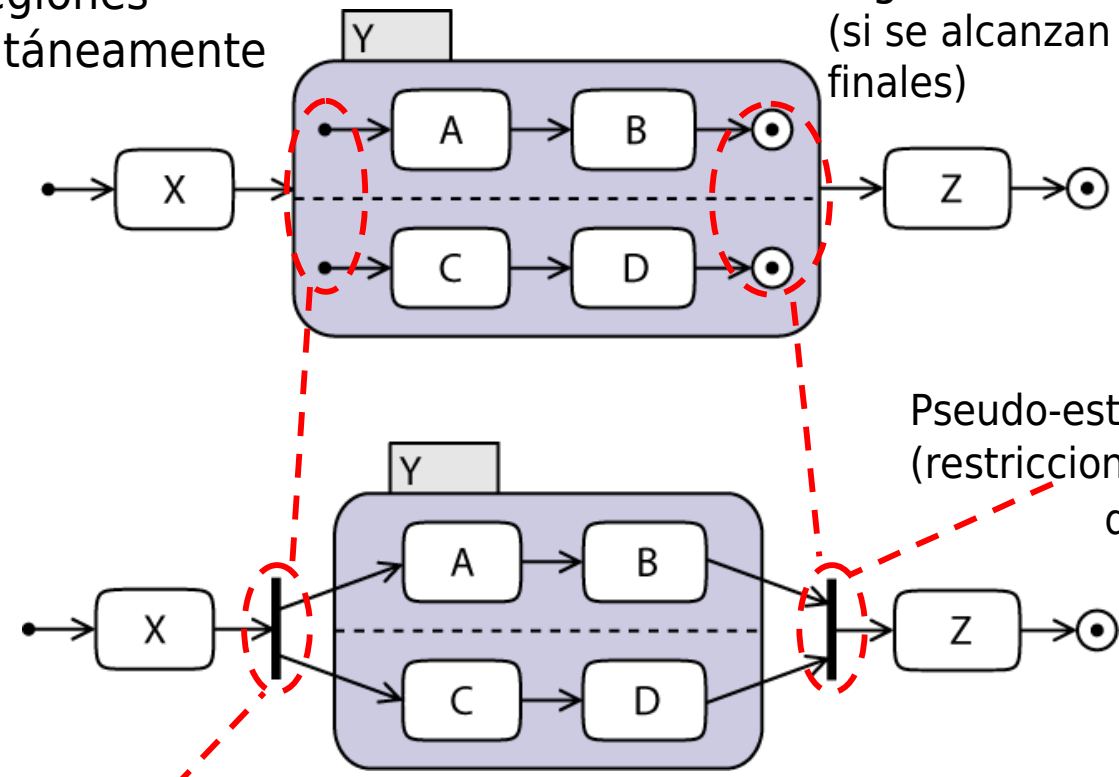
regiones ortogonales, ambas activas si el Client/Server está activo

Los estados ortogonales son “concurrentes” ya que un único evento puede disparar una transición en cada región ortogonal “simultáneamente”

6 – Máquinas de estados
Forks y *joins*

se entra en todas las regiones simultáneamente

se abandonan todas las regiones simultáneamente (si se alcanzan los estados finales)

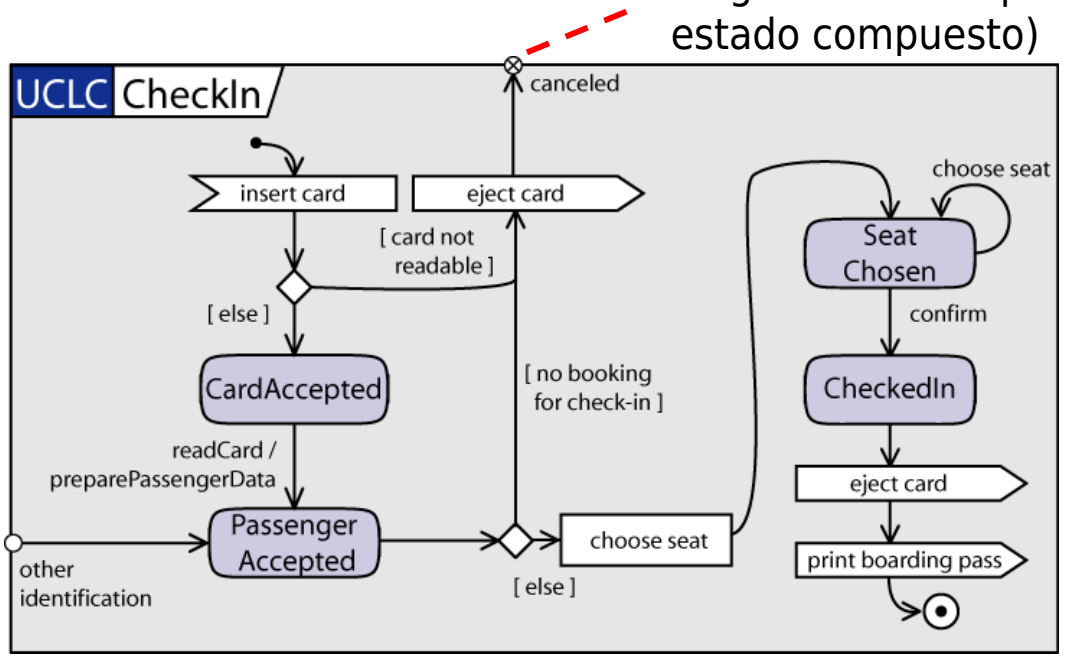


Pseudo-estado *join* (restricciones duales a los de los *fork*)

Pseudo-estado *fork*
(una transición entrante, al menos dos salientes; las transiciones salientes deben alcanzar estados de diferentes regiones de un estado ortogonal)

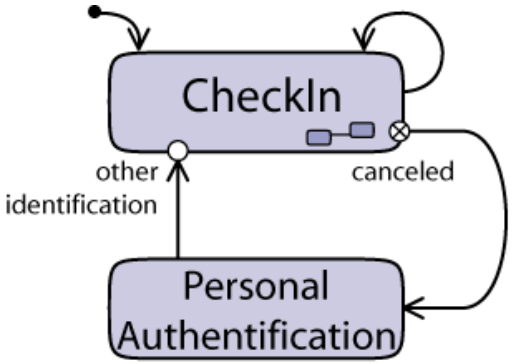
6 – Máquinas de estado
Puntos de entrada y salida (1)

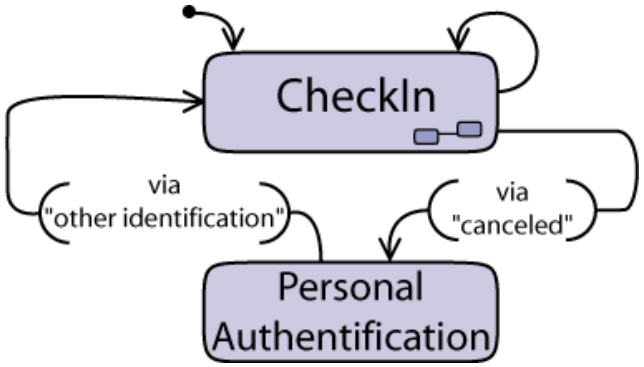
- Puntos de entrada y salida (tipo de pseudo-estado)
  - proporcionan mejor encapsulación de estados compuestos
  - ayuda a evitar transiciones “desestructuradas”



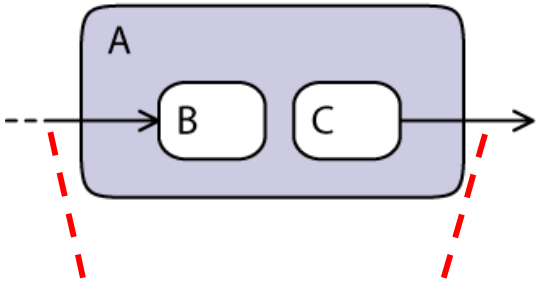
6 – Máquinas de estado
Puntos de entrada y salida (2)

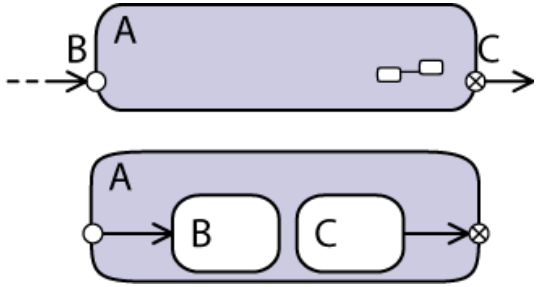
Alternativas notacionales





Semánticamente equivalentes





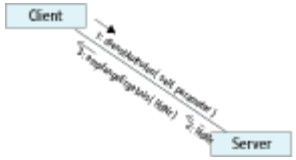
Transiciones “desestructuradas”

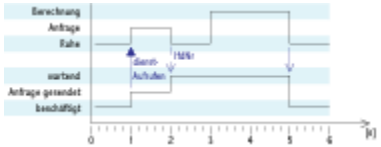
6 – Máquinas de estados
Conclusiones

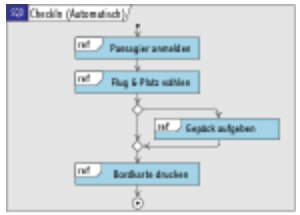
- Las máquinas de estados modelan comportamiento
  - ciclos de vida de objetos y casos de uso
  - autómatas de control
  - protocolos
- Las máquinas de estados constan de
  - regiones y …
  - … (pseudo-)estados (con actividades *entry*, *exit* y *do*) …
  - conectadas mediante transiciones (con disparadores, guardas, y efectos)
- Las máquinas de estados se comunican via pools de eventos.
- Las máquinas de estados se ejecutan mediante pasos *run-to-completion*.

Unified Modeling Language 2

Parte 7 - Interacciones







7 - Interacciones
Un primer vistazo

- Tres presentaciones semánticamente equivalentes (?):

diagrama de secuencias

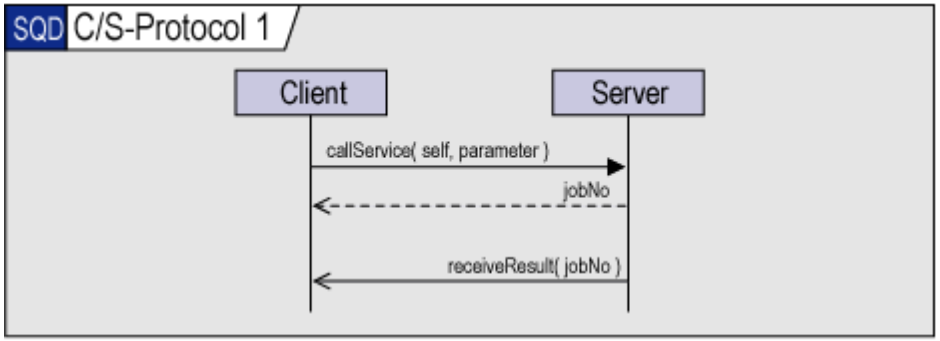


diagrama de comunicación

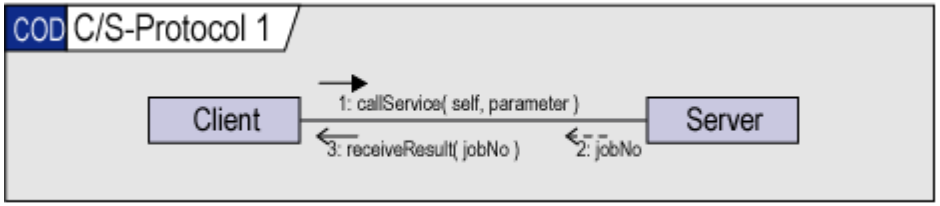
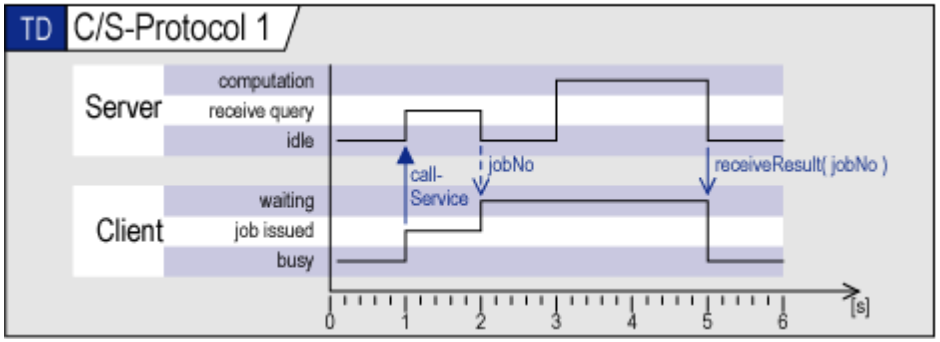
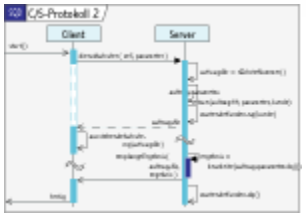


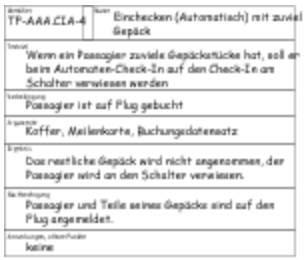
diagrama de temporización

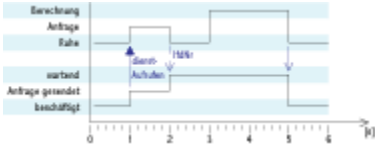


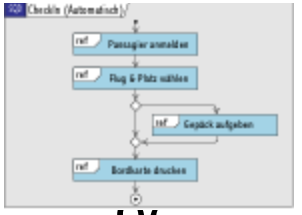
7 - Interacciones
Escenarios de uso

- Interacciones de clase/objeto
  - diseño/documentación de intercambio de msj entre objetos
  - expresión de mensajes síncronos/asíncronos, señales y llamadas, activación, restricciones de temporización
- Escenarios de casos de uso
  - ilustración de caso de uso con un escenario concreto
  - útil para el diseño/documentación de procesos de negocio (esto es, fase de análisis y reingeniería)
- Casos de prueba
  - descripción de casos de prueba en todo nivel de abstracción
- Especificación/documentación de temporización
- *Interaction overview*
  - estilo más visual para representar gran nº de interacciones
  - definido equivalente al uso de operadores de interacción





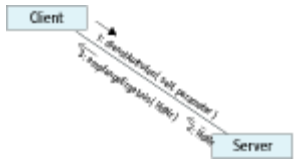




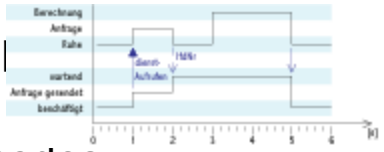
7 - Interacciones
Variantes sintácticas

- Diagrama de secuencias
  - el tradicional + operadores de interacción
  - focaliza en el intercambio de muchos mensajes en patrone complejos de entre unos pocos *partners* de interacción

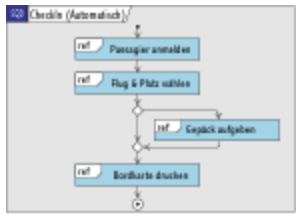
- Diagrama de comunicación
  - “diagrama de colaboración” en UML 1.x
  - focaliza en el intercambio de unos pocos mensajes entre (muchos) *partners* en configuración compleja



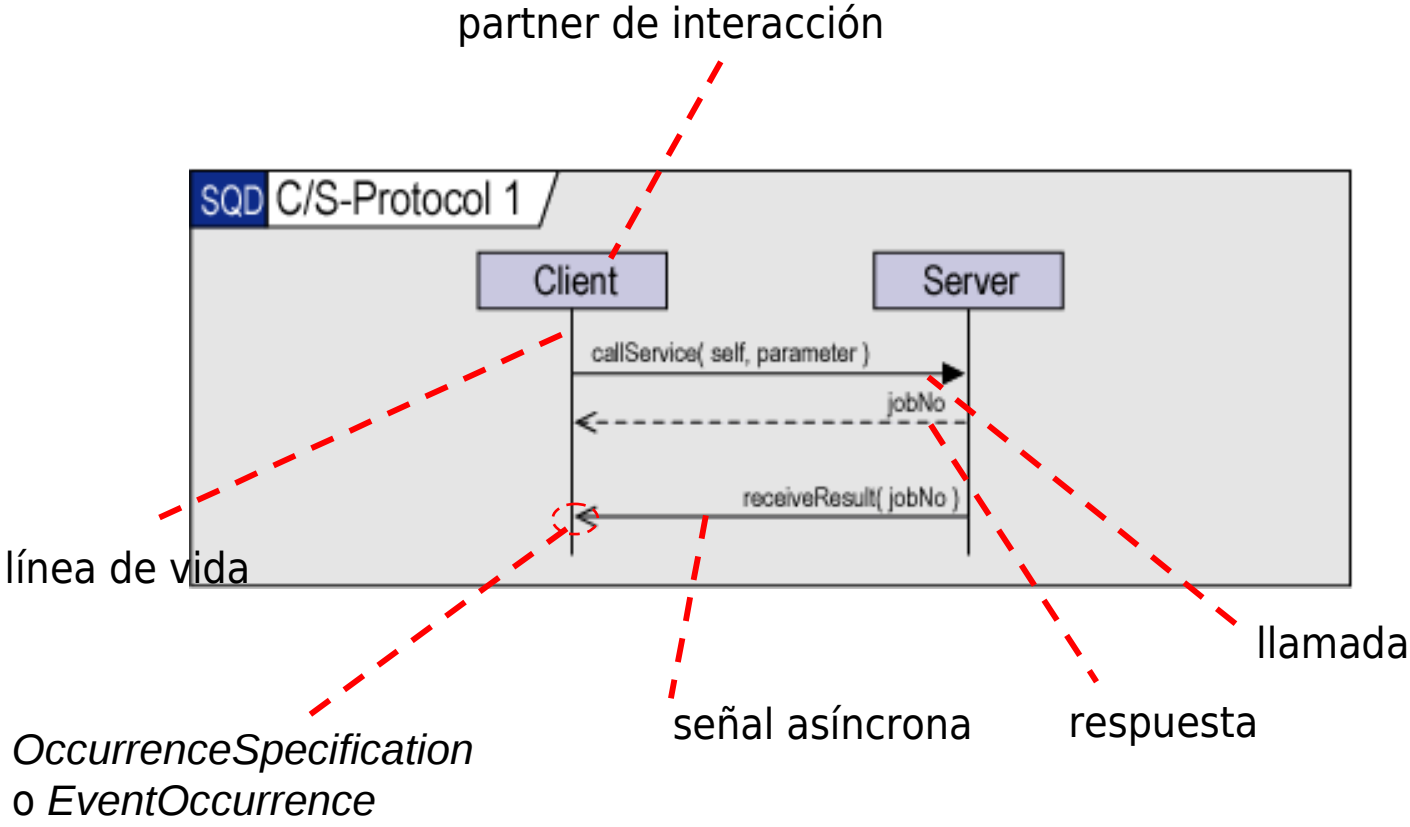
- Diagrama de temporización
  - nuevo en UML 2, representación tipo osciloscopio, tiem necesariamente métrico
  - focaliza en el tiempo (real) y cambios de estado coordinados de los *partners* en el tiempo



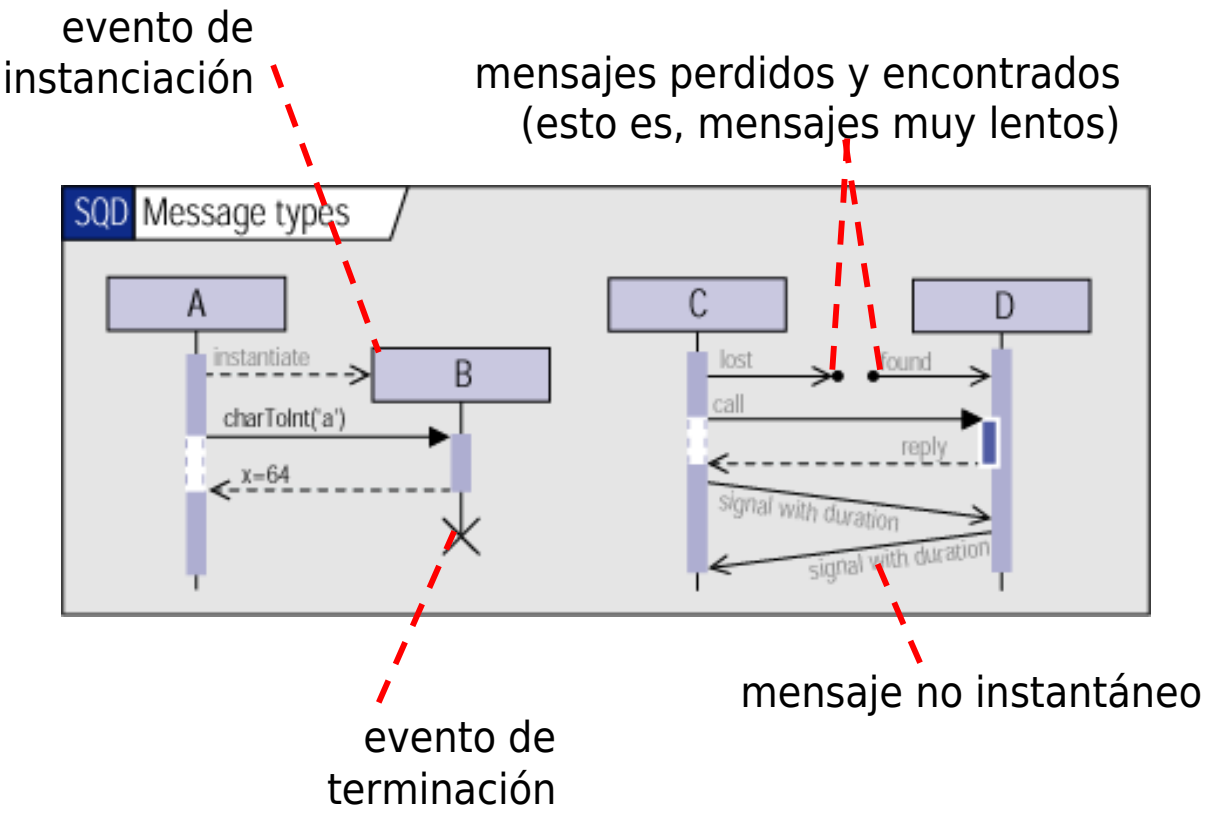
- Diagrama de *interaction overview*
  - parece un diagrama de actividades restringido, pero no lo
  - dispone interacciones elementales para destacar su organización



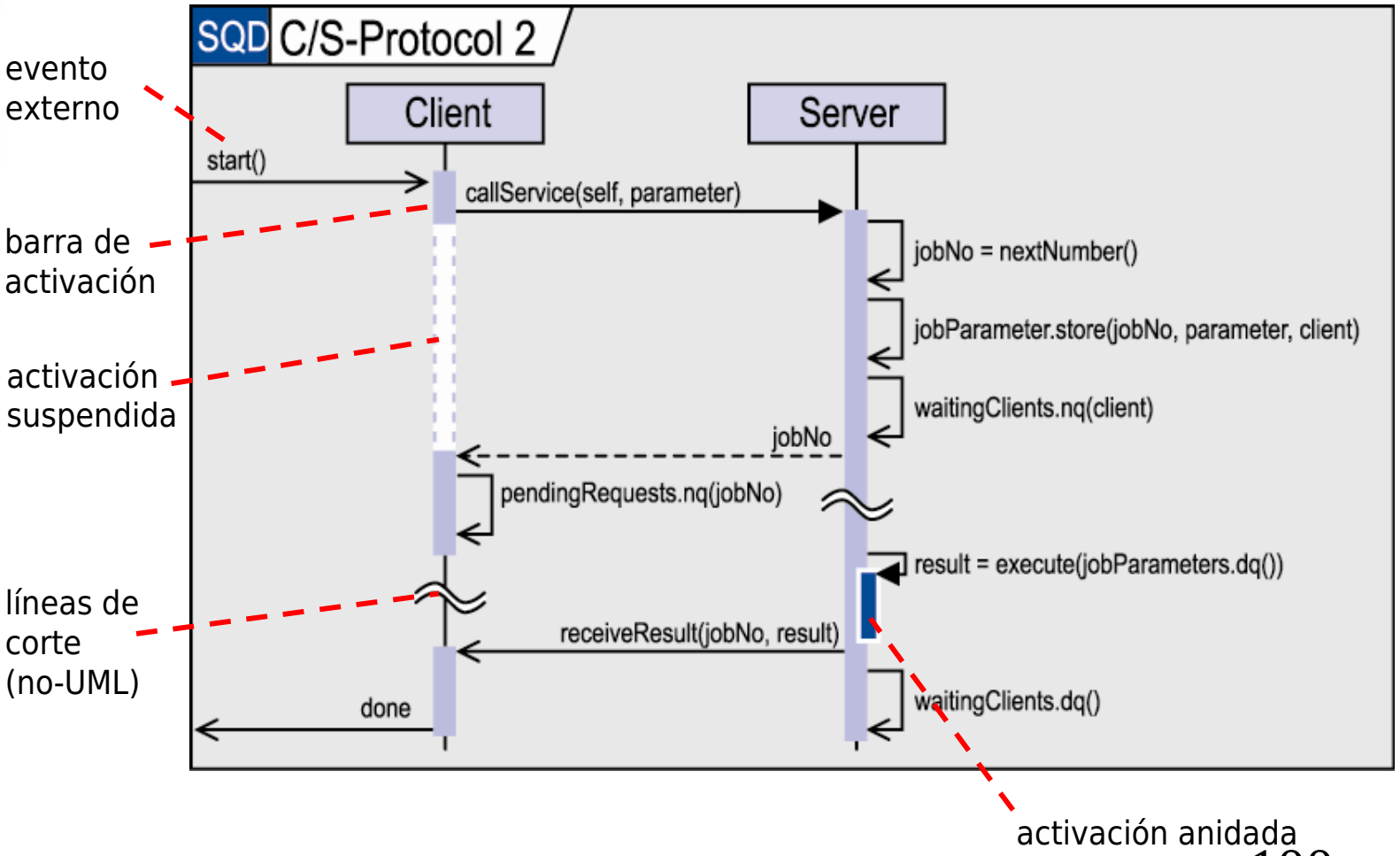
7 – Interacciones simples
Conceptos principales



7 – Interacciones simples
Tipos de mensajes

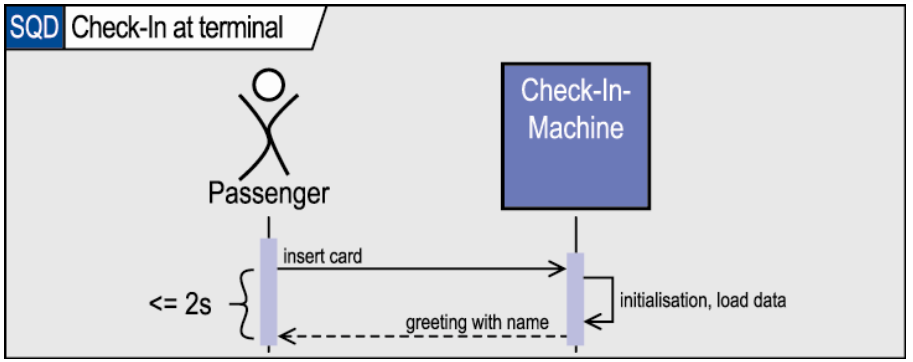


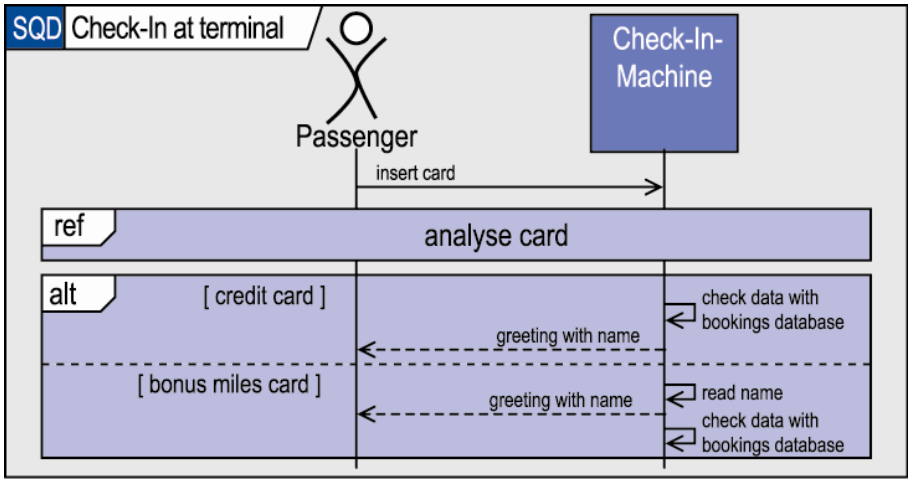
7 – Interacciones simples
Activación



7 - Interacciones
Uso: Escenarios de casos de uso

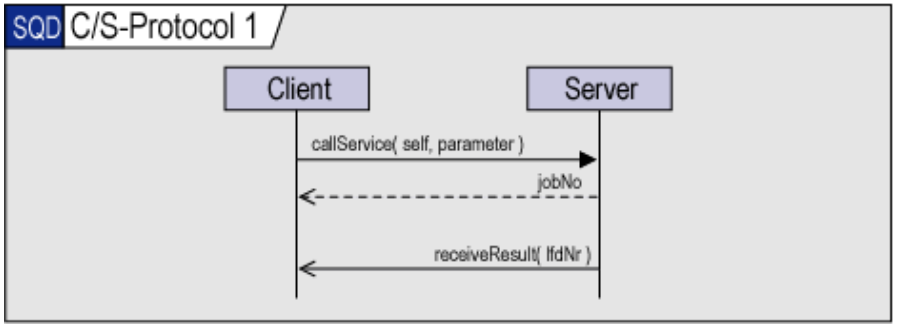
- Las participantes en una interacción son actores y sistemas (en lugar de clases y objetos).
- Pueden refinarse sucesivamente.
- Útil también para especificar requisitos no funcionales de alto nivel tales como tiempos de respuesta.
- Dependiendo de las circunstancias puede aplicarse todo tipo de diagramas de interacción.

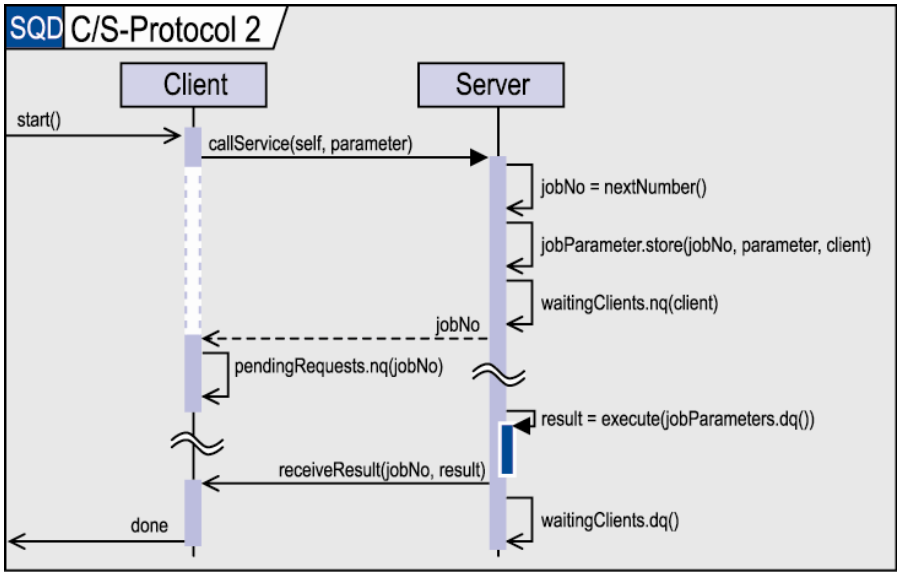




7 - Interacciones
Uso: Interacciones entre clases

- Los participantes en una interacción son clases y objetos (en lugar de actores y sistemas).
- De nuevo pueden aplicarse sucesivos refinamientos de diferentes maneras:
  - descomposición de mensajes
  - descomposición de la estructura de los participantes en la interacción.
- Puede aplicarse todo tipo de diagrama de interacción, dependiendo de las circunstancias.





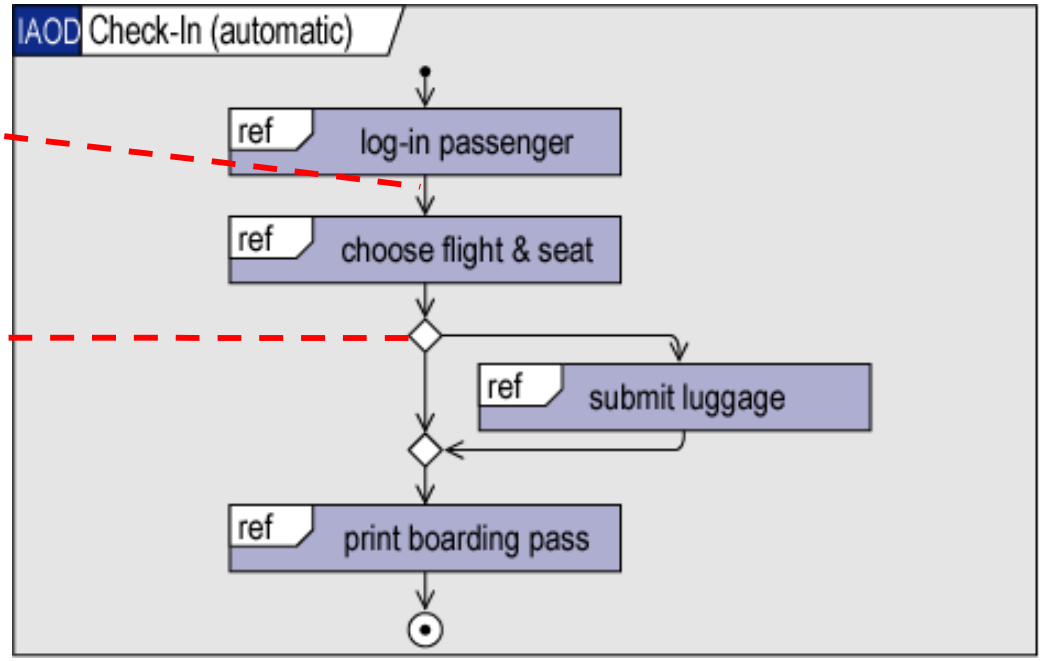
7 - Interacciones
Uso: *Interaction overview*

- Organizan gran nº de interacciones en un estilo + visual
- Definidas como equivalentes al uso de operadores de interacción en diagramas de secuencia básicas

secuenciación, probablemente equivalente al operador *seq*

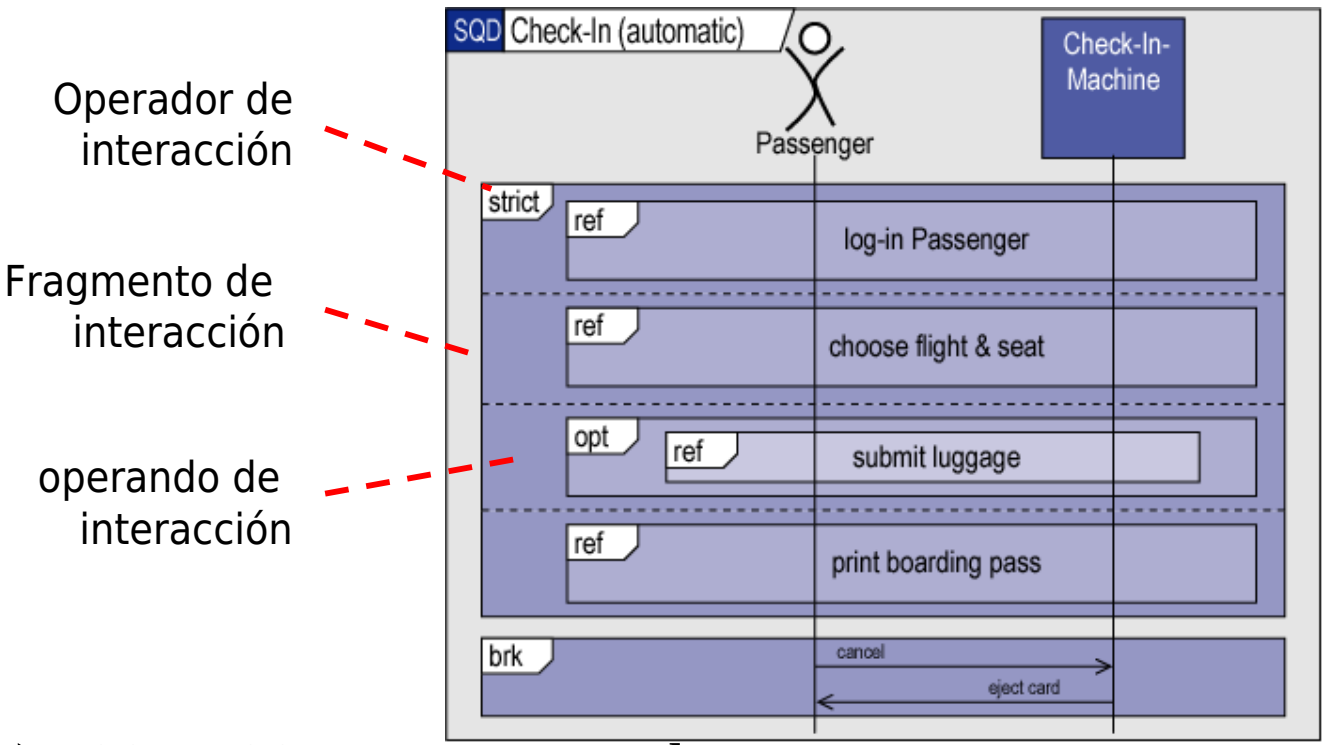
choice/merge, equivalente a *alt/opt*

también permitido: *fork/join* (se dice equivalente a *par,* pero ...)



7 - Interacciones
Interacciones complejas

- Una interacción compleja es como una expresión funcional:
  - un *InteractionOperator*,
  - uno/varios *InteractionOperands* (separados por líneas discontinuas),
  - (y a veces también números o conjuntos de señales).



7 – Interacciones
Operadores de interacción (visión general)

- *strict*
  - secuenciamiento por operandos
- *seq*
  - secuenciamiento por línea de vida
- *loop*
  - *seq* repetido
- *par*
  - entrelazamiento de eventos
- *region* (crítica)
  - suspende el entrelazado
- *consider*
  - restringe el modelo a mensajes especificos (esto es, permite cualquier otro mensaje en cualquier sitio)
- *ignore*
  - dual a consider
- *ref*
  - macro-expansión del fragmento
- *alt*
  - ejecución alternativa
- *opt*
  - ejecución opcional
  - azúcar sintáctico para *alt*
- *break*
  - aborta la ejecución
  - a veces escrito como “brk”
- *assert*
  - elimina incertidumbre en la especificación (esto es, declara toda traza válida)
- *neg*
  - declara toda traza inválida (→ semántica tri-valuada)

7 - Interacciones
Conclusiones

- Adición de interacciones complejas parecidos a los MSC de alto nivel.
- Nuevos tipos de diagramas:
  - diagramas de temporización (tipo osciloscopio), y
  - *interaction overview* (similar a un diagrama de actividad restringido)
  - diagrama de colaboración renombrado como diagrama de comunicación
- Metamodelo completamente nuevo.
- Semántica tri-valuada casi formal (???) con trazas de eventos entrelazados válidas, inválidas e inconcluyentes.
- Algunos problemas semánticos aún no resueltos.

Unified Modeling Language 2

Part 8 – Observaciones finales

Software de Comunicaciones 2007-2008

8 – Observaciones finales
Gestión de modelos grandes

- Gestionar modelos grandes es un problema en sí:
  - gestión de versiones, *diff*/*patch*, *merge*
  - migración entre herramientas
  - mantenimiento de modelos a largo plazo
  - *round-trip* (ida y vuelta) con interferencia manual
  - medidas, consultas, comprobaciones, análisis de modelos
  - simulación, generación de código
  - documentación
- Hoy en día, no se dispone de herramientas de soporte para la mayoría de estas tareas.
- Es muy engorroso hacerlas a mano.

8 – Observaciones finales
Experiencias industriales

- En contra de la creencia habitual, muchos expertos del dominio se muestran bastante cómodos ante los diagramas UML, sólo del nivel de análisis, claro.
- Con UML 2, es posible captar ahora muchas cosas que antes resultaban difíciles de captar.
- El soporte de las herramientas aún es insuficiente, no obstante, en parte debido a la enorme complejidad del UML.
- Por último: se trata de un paso hacia delante, pero aún no hemos llegado.

8 – Observaciones finales
Una mirada a la bola de cristal

- Es bastante probable que sobrevenga una versión UML 2.2 para tratar los problemas actualmente presentes en el UML.
- Habrá probablemente un UML 2.3 y un UML 2.4 también – pero, ¿habrá un UML 3.0?
- Sólo puede haber un lenguaje de modelado *unificado*, aunque habrá, probablemente, lenguajes de modelado más simples.
- Los lenguajes específico a un dominio ni están unificados ni (usualmente) son más simples que el UML y apenas se han visto en ningún sitio evidencias de sus pretendidos beneficios.