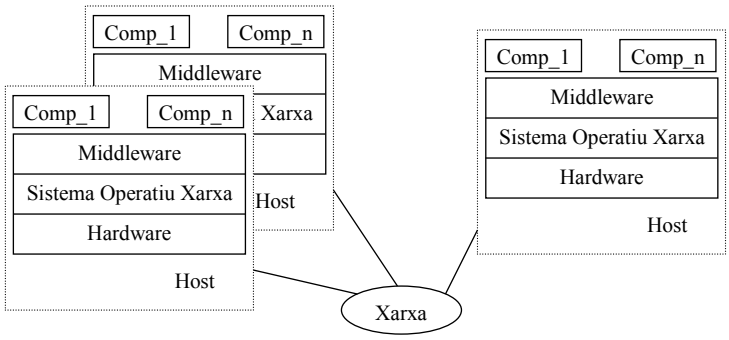
Disseny de sistemes distribuïts

- Introducció
- Mecanismes de comunicació
- Tecnologies de sistemes distribuïts
- Tipus de sistemes distribuïts
- Alguns patrons de disseny distribuït
  - Patrons Façana Remota i DTO
- Arquitectures orientades a serveis (SOA)
  - Conceptes generals
  - Patró Localitzador de Serveis
  - Disseny de serveis
  - Disseny basat en serveis

Sistemes distribuïts

Un sistema distribuït és una col.lecció de nodes autònoms connectats mitjançant una xarxa. Cada node executa components, i usa un *middleware* que permet coordinar les activitats dels components de manera que els usuaris veuen el sistema com un únic recurs informàtic integrat.



Sistemes centralitzats vs. distribuïts

- Sistemes centralitzats:
  - No tenen components autònoms.
  - Homogenis.
  - Molts usuaris comparteixen el sistema contínuament.
  - Un únic punt de control i de fallada.
- Sistemes distribuïts:
  - Tenen components autònoms.
  - Heterogenis.
  - No tots els usuaris usen el mateix component.
  - Concurrència.
  - Diversos punts de fallada.

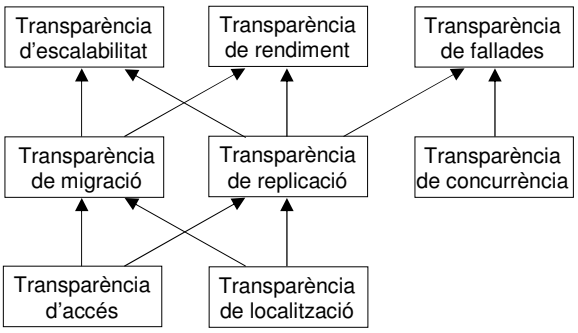
Característiques dels sistemes distribuïts

- Avantatges:
  - Compartició de recursos.
  - Apertura (*openness*).
  - Concurrència.
  - Escalabilitat.
  - Tolerància a fallades.
- Inconvenients:
  - Alta complexitat.
  - Disminució de la seguretat.
  - Esforç de gestió.
  - Impredictibilitat.

Transparències en un sistema distribuït

Objectiu: el fet que un sistema sigui distribuït hauria de ser transparent

Hi ha diverses dimensions de transparència:



Causes de la distribució

- Algunes causes:
  - Millorar el rendiment i/o l'administració (de part) del sistema
    - ◊ p.e., minimitzar els accessos a dades remotes replicant-les
  - Adaptar-se a la funcionalitat que proporciona un node
    - ◊ p.e., localització de les dades
  - Adaptar-se a la organització
    - ◊ p.e., localització del departament d'administració, organització multi-site
  - Condicionaments tecnològics
    - ◊ p.e., capa de presentació web, ús de tecnologies de serveis Web
  - Condicionaments d'administració
    - ◊ p.e., les còpies de seguretat s'han de fer en un node determinat
  - Ús de software existent
    - ◊ p.e., integració de un paquet comercial (ERP, …), integració amb un sistema heredat (*legacy*), accés a serveis d'un proveïdor

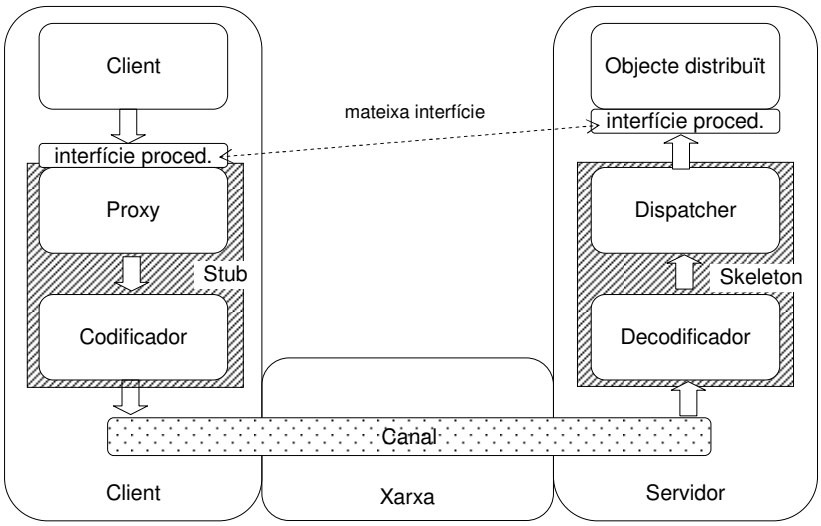
Middleware

- Un *middleware* és una col·lecció de components d'infraestructura que permeten als components de domini d'un sistema comunicar-se l'un amb l'altre a través d'una xarxa o d'un sistema complex, possiblement sobre diverses plataformes
- El middleware proporciona dos tipus de serveis:
  - de desenvolupament: destinats a facilitar el desenvolupament de les aplicacions
    - Exemple: facilitar la integració de diverses fonts de dades de manera transparent
  - d'administració: destinats a facilitar l'administració dels sistemes distribuïts
    - Exemple: activar i desactivar components a un node
- Els mecanismes de comunicació més importants són:
  - RPC (*Remote Procedure Call*)
  - MOM (*Message-Oriented Middleware*)

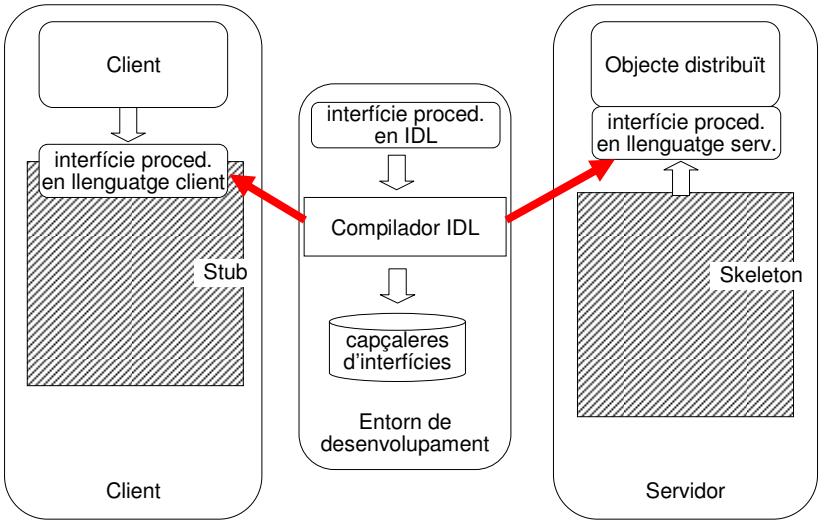
RPC vs. MOM

- Middleware RPC
  - Es basa en el patró *Remote Proxy*
  - El client crida a una operació com si fos una crida local i el middleware s'encarrega d'executar-la al servidor
  - Pot aprofitar per donar serveis addicionals (seguretat, transaccions, etc)
- Middleware MOM
  - El client publica un missatge a una cua de missatges
  - El servidor recull els missatges i reacciona adequadament
  - Arquitectures més desacoblades

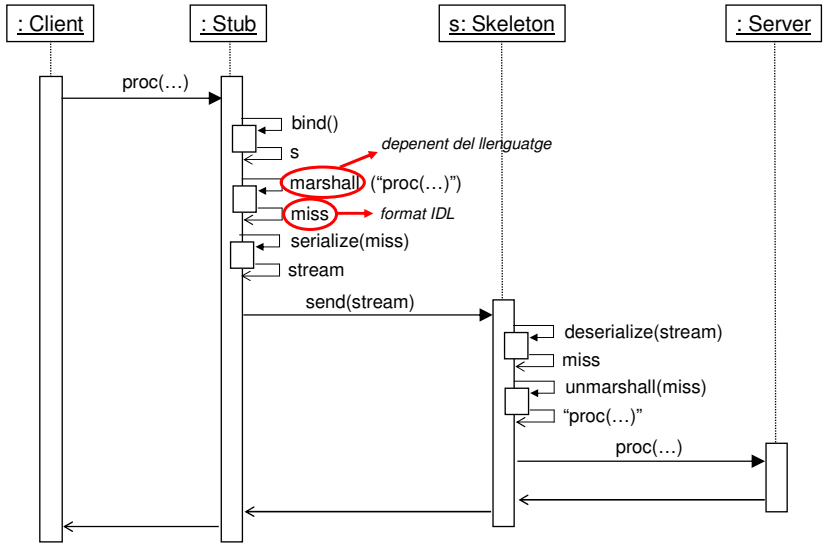
RPC: Estructura Interna



RPC: Preparació



RPC: Crida



Stub i skeletons controlen detalls de baix nivell (timeouts, retransmisions, etc.)

Tecnologies de components distribuïts

- El disseny de sistemes distribuïts orientats a objectes ha de conjugar:
  - la resolució del problema adreçat (aspectes del domini)
  - el proveïment de diversos serveis transversals d’interès (persistència, transaccions, seguretat, eficiència, etc.)
- Les tecnologies de components distribuïts possibiliten el disseny de sistemes distribuïts mantenint separats aquests dos aspectes:
  - el dissenyador es pot focalitzar en els aspectes del domini
  - la tecnologia emprada proveeix els serveis transversals
- Tecnologies de components distribuïts principals:
  - J2EE/EJB: Enterprise Java Beans
  - .NET de Microsoft
  - Web Services
  - CCM: CORBA Component Model
  - …

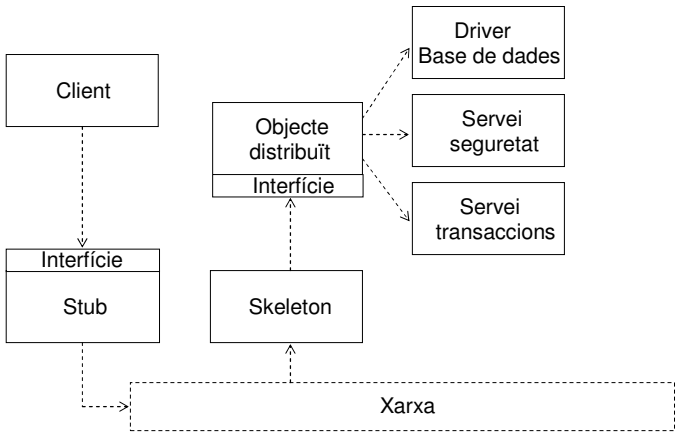
Components i Contenidors

Les aplicacions consten de *components* que normalment s'executen en el context de *contenidors*

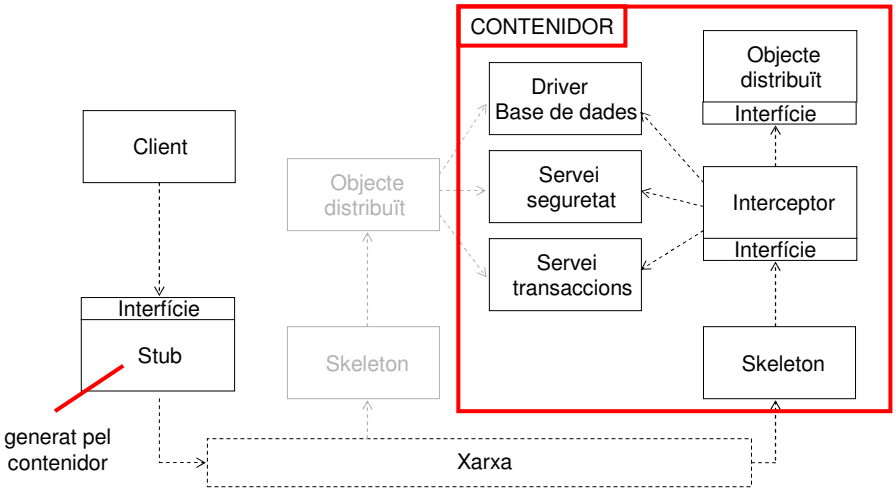
Un component és una unitat atòmica des del punt de vista de gestió i desplegament:
- Es comunica amb d'altres components mitjançant les seves interfícies
- Consta de classes, fitxers diversos (descriptors, ...) i altres possibles recursos
- Pot ser de diverses menes (negoci, web, ...)

Un contenidor forma l'entorn controlat en què s'executa un component:
- Aïlla el component dels seus clients (i vice-versa)
- Ofereix serveis transversals als components

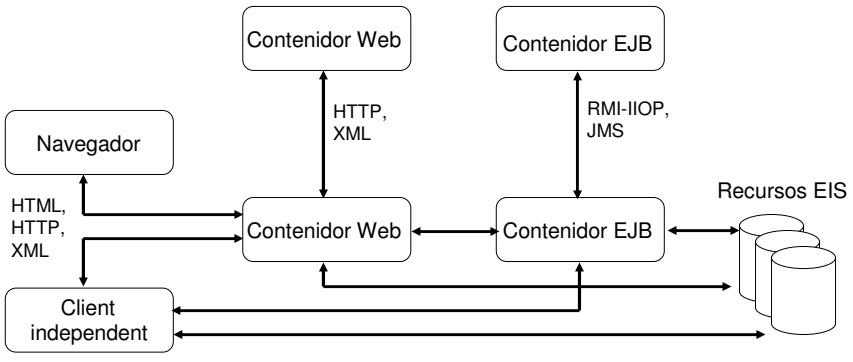
Paper del contenidor en una arquitectura distribuïda
Arquitectura sense contenidor



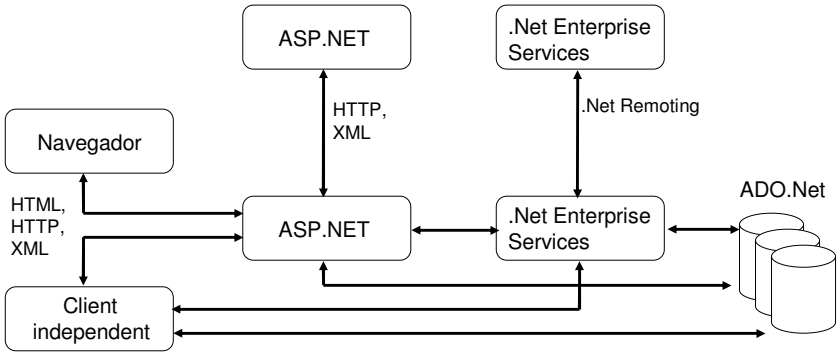
Paper del contenidor en una arquitectura distribuïda
Arquitectura amb contenidor



Tecnologia J2EE



Tecnologia .NET



Fronteres de distribució

Anomenem fronteres de distribució a la separació física entre els components de l'arquitectura del sistema

En una arquitectura en tres capes, les fronteres de distribució poden:

- Assignar capes diferents a nodes diferents, i/o
- Assignar capes diferents a un mateix node (fins i tot, fusionades en un mateix component)
- Distribuir una mateixa capa entre més d'un node

Les fronteres de distribució són el resultat de les vistes de desenvolupament i desplegament del sistema (v. Tema "Patró Arquitectura en Capes"):

- La vista de desenvolupament determina els components de l'arquitectura
- La vista de desplegament assigna aquests components a elements del hardware

Les fronteres de de distribució determinen el tipus de sistema distribuït

Tipus de sistemes distribuïts

Algunes fronteres habituals són:

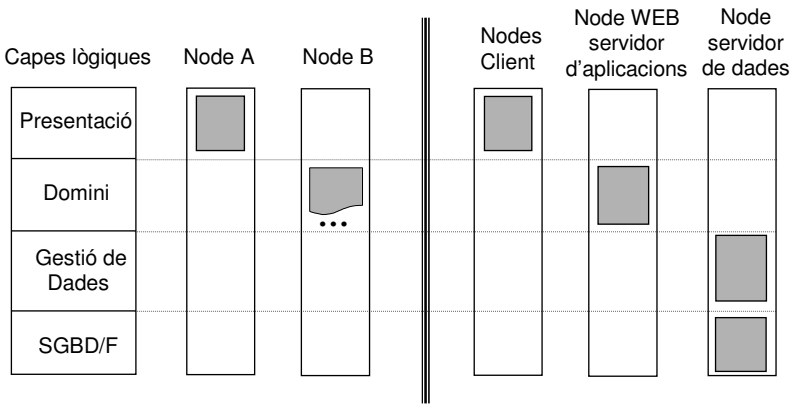
- Presentació remota → típic sistema RPC
- Presentació distribuïda → típic sistema Web
- Dades remotes → protocol SGBD
- Dades distribuïdes → distribució ad-hoc (e.g., actualització de 2 BDs); replicació
- Distribució de la capa de domini → típic sistema SOA

Freqüentment dins un mateix sistema distribuït es tracen diverses fronteres

També trobem algunes situacions que condicionen el traç de les fronteres. p.e.:

- Integració de sistemes heretats
- Integració de sistemes o serveis de tercers

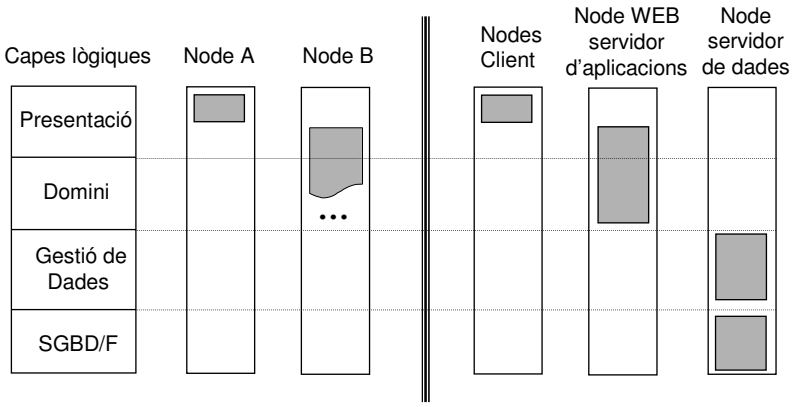
Presentació remota



Cas general

Exemple: *rich client*

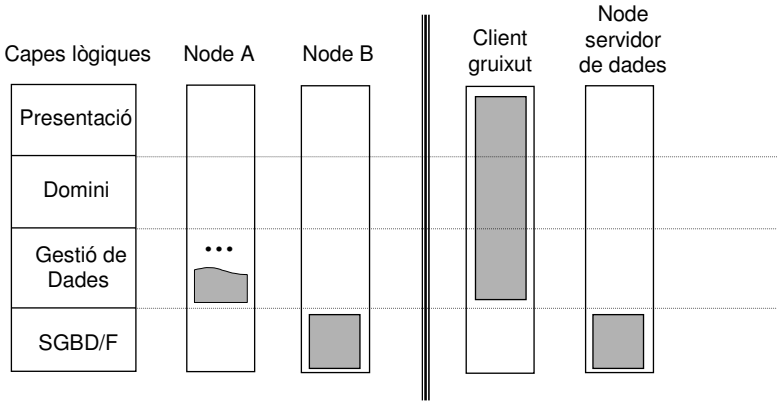
Presentació distribuïda



Cas general

Exemple: *thin client*

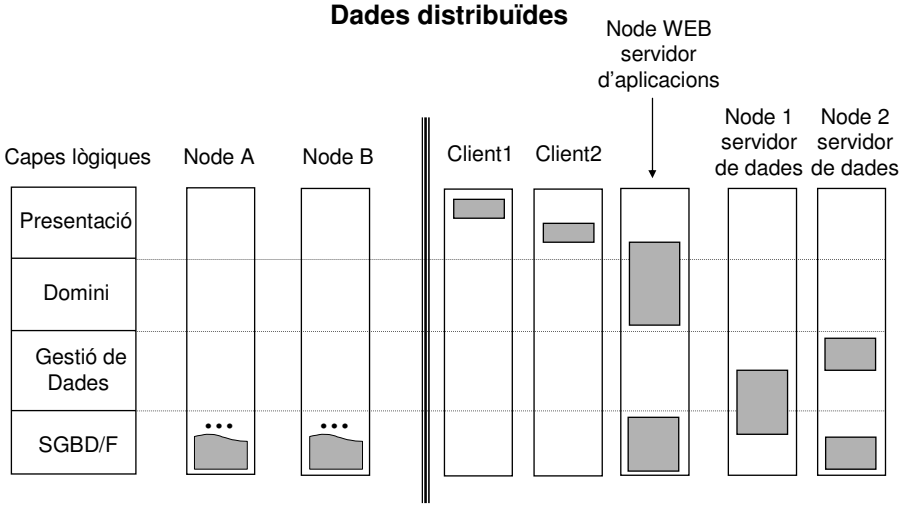
Dades remotes



Cas general

Exemple: *thick client*

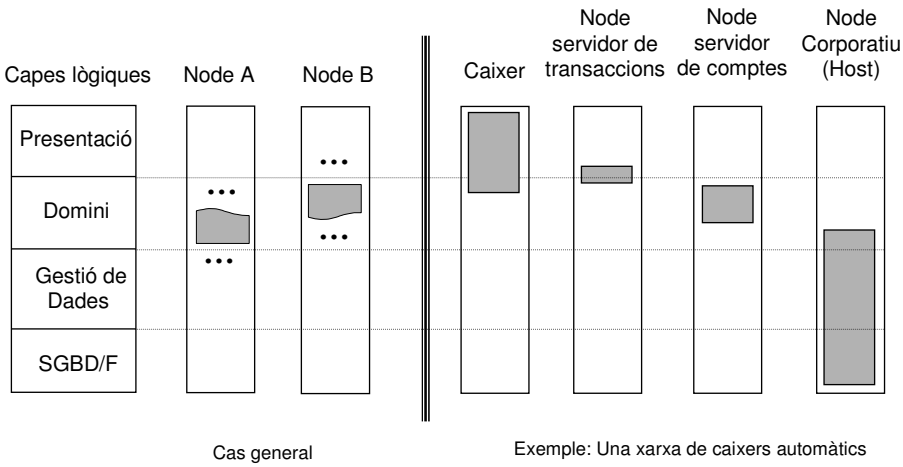
Dades distribuïdes



Cas general

Exemple: sistema amb distribució de capes de presentació i dades

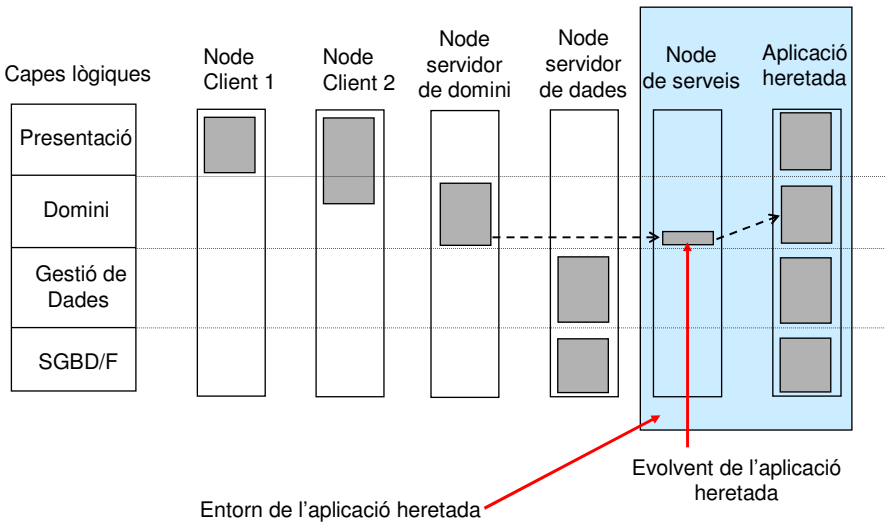
Domini distribuït



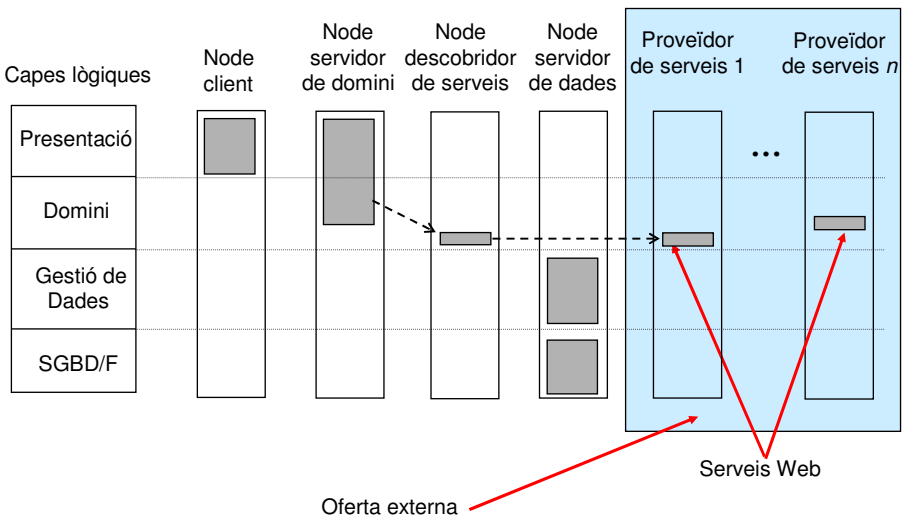
Cas general

Exemple: Una xarxa de caixers automàtics

Exemple: integració de sistemes heretats
Utilització de tècniques d'evolvent



Exemple: integració de sistemes de tercers
Serveis Web



Alguns patrons de disseny distribuït

- Una crida remota és diversos ordres de magnitud més costosa que una crida local
  - transformacions (binding+marshaling+serialize i vice-versa)
  - altres: comprovacions permisos, enrutament, transmissió pròpiament dita, …
- Per tant, la interfície que fem servir per a invocar localment un objecte ha d'obeir uns principis diferents de la que fem servir per invocar-lo localment
  - La interfície local normalment serà de granularitat fina mentre que la remota ha de ser de granularitat gruixuda
- Primera llei dels objectes distribuïts: No distribueixis els teus objectes!
- Patrons de disseny per al disseny d'interfícies remotes:
  - Patró Façana Remota (*Remote Facade*)
  - Patró *Data Transfer Object* (DTO)
  - Ambdós patrons s'acostumen a aplicar junts
- A més, patró Localitzador de Servei (Service Locator), més endavant

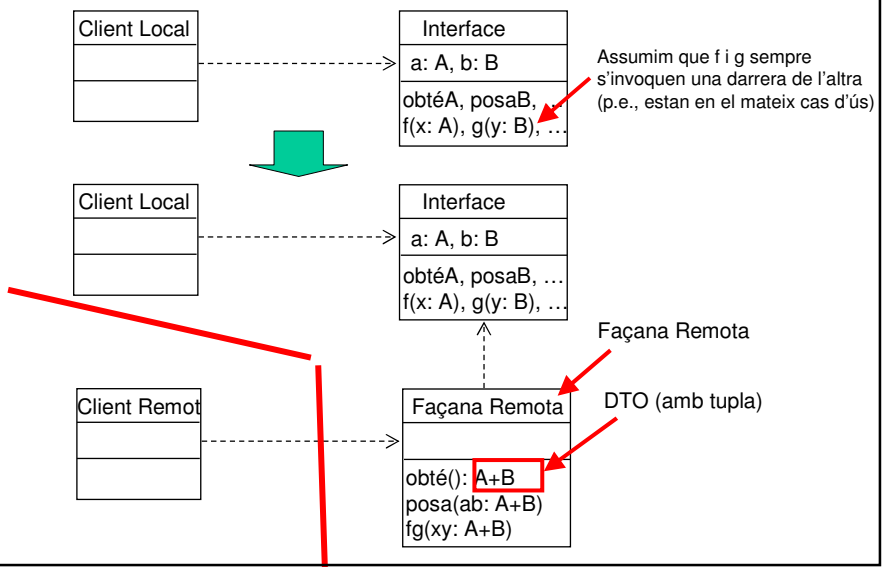
Patró Façana Remota

- **Context**
  - Es disposa d'una interfície d'objecte de granularitat fina
    - aquesta interfície satisfà diversos principis i propietats
  - L'objecte pot formar part d'un sistema distribuït
- **Problema**
  - Utilitzar la interfície de granularitat fina per a accessos remots és costós
  - Volem mantenir la interfície de granularitat fina per als accessos locals
- **Solució**
  - Afegir una nova classe (la façana remota) amb mètodes de granularitat més gruixuda que faci servir la interfície local per implementar la funcionalitat

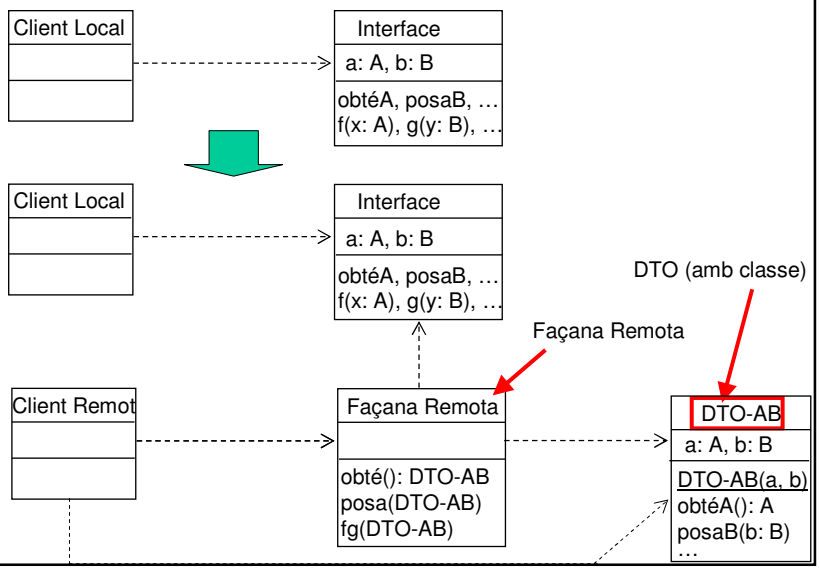
Patró DTO

- **Context**
  - Estem definint la interfície remota d'un objecte (segurament per l'aplicació del patró Façana Remota)
- **Problema**
  - Com que volem reduir el número de crides que fem, cada crida ha de portar més informació als paràmetres i al valor de retorn
  - La llista resultant pot ser molt llarga
- **Solució**
  - Definim un nou grup de dades que contingui totes les dades que s'han de passar com a paràmetre o com a resultat. Els objectes que són agrupacions d'aquesta mena s'anomenen *Data Transfer Objects* (DTO)
  - Aquesta agrupació es pot definir com una classe (classe DTO) amb operacions *getter* i *setter*, o bé com una tupla, si el llenguatge ho soporta

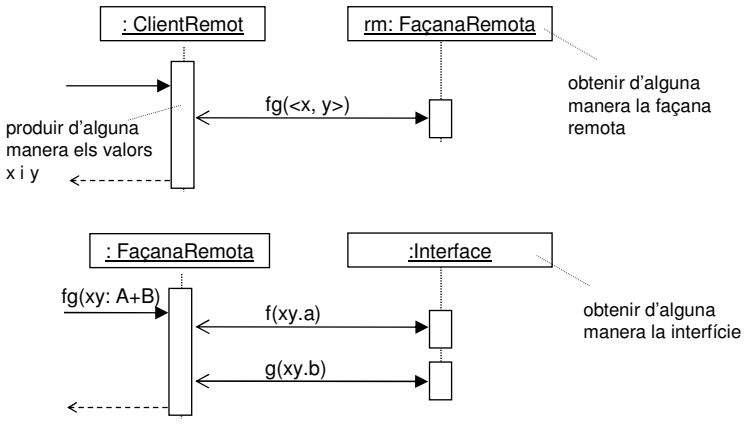
Patrons Façana Remota + DTO
Aspecte estàtic, DTO amb tupla



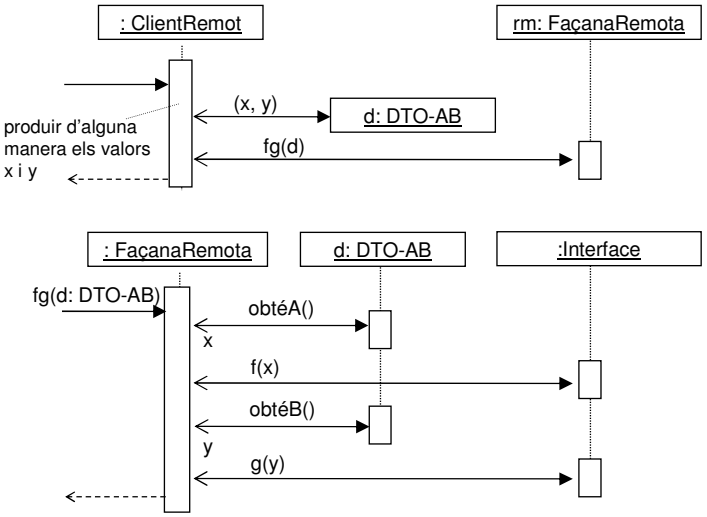
Patrons Façana Remota + DTO
Aspecte estàtic, DTO amb classe



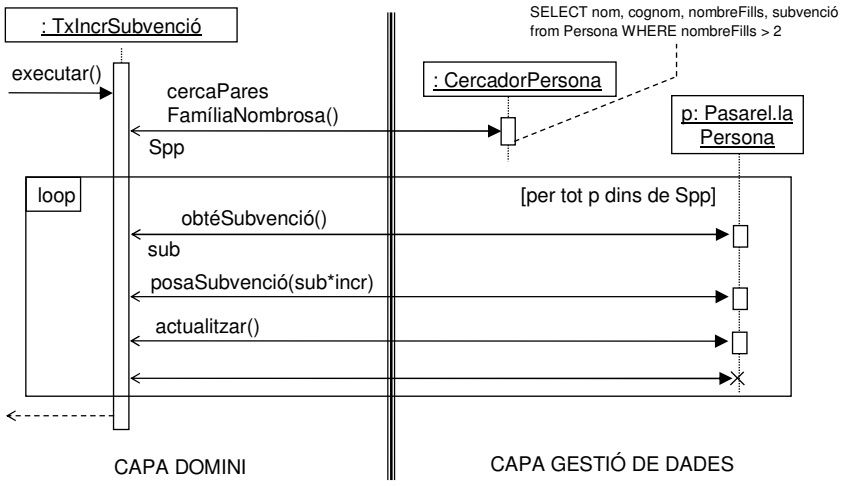
Patrons Façana Remota + DTO
Aspecte dinàmic, DTO amb tupla



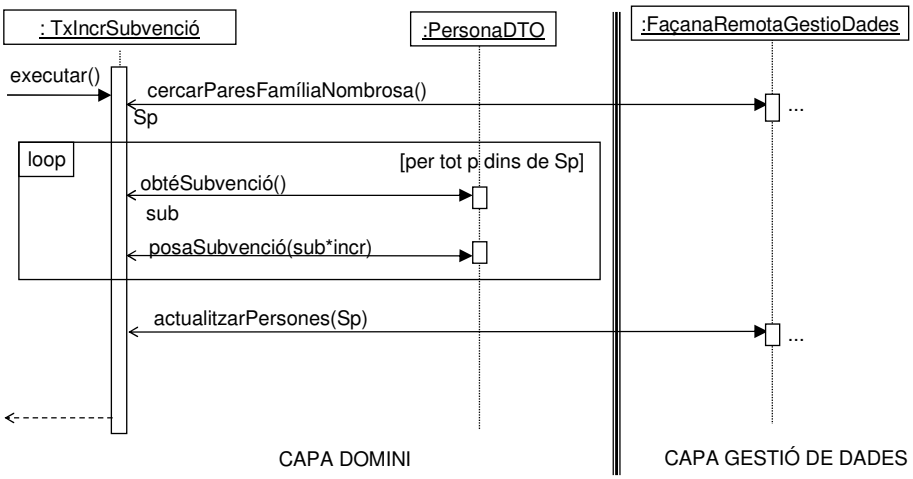
Patrons Façana Remota + DTO
Aspecte dinàmic, DTO amb classe



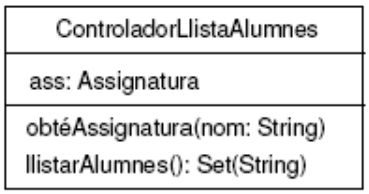
Exemple 1
Disseny centralitzat amb capa de dades mitjançant passarel·la fila



Exemple 1
Distribució de les capes de domini i gestió de dades en nodes diferents



Exemple 2
Disseny amb controladors de cas d'ús a la capa de domini



Volem distribuir la capa de presentació i la capa de domini en nodes diferents.
Cal aplicar algún patró d'interfícies remotes?
– Patró façana remota: volem introduir una interfície de granularitat gruixuda?
  ◊ aquest controlador de cas d'ús ja té granularitat gruixuda
– Patró DTO: volem introduir més informació als paràmetres?
  ◊ el Set(String) ja porta tota la informació que cal

En aquest cas, el disseny centralitzat ens ha portat a unes interfícies de granularitat gruixuda i objectes tipus DTO per a la comunicació de les capes de presentació i domini

Arquitectures orientades a serveis (SOA)

- Construcció d'aplicacions a partir de serveis
  - desenvolupats a mida
  - ja existents → permet integrar sistemes ja construïts
- Evolució del paradigma clàssic de programació modular
  - descomposició d'un problema en unitats
  - col.lectivament, les unitats representen una peça d'ún procés de negoci
  - individualment, les unitats són inpendents i poden distribuir-se
- Permet independitzar un servei del seu proveïdor
  - el client només coneix quin servei vol demanar i com demanar-lo

Exemple

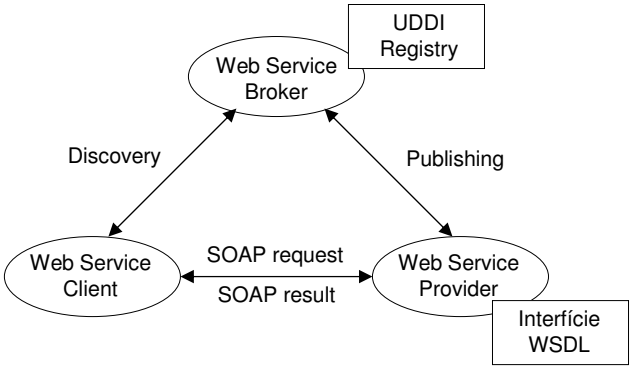
- Companyia que disposa de:
  - sistema d'inventari
  - sistema de comptabilitat
  - sistema de gestió de la relació amb el client (CRM)
  - sistema de gestió de magatzems
- Quan es produeix una venta cal:
  - actualitzar l'inventari del producte venut
  - introduir els canvis en el *cash flow* del sistema de comptabilitat
  - obtenir les dades del client del CRM
  - obtenir les dades de la situació del producte al magatzem per al seu enviament
  - possiblement, realitzar una comanda a un proveïdor

Conceptes bàsics de SOA

SOA es basa en tres rols:

- Service Provider
  - proporciona serveis, els descriu i els publica
- Service Requestor
  - obté un Service Provider adequat, proporcionat per un Service Broker
  - invoca el servei corresponent
- Service Broker
  - gestiona una taula de descripcions de serveis
  - enllaça els Service Provider amb els Service Requestor

Exemple: rols SOA en la tecnologia Web Services



Què és un servei

- Components que encapsulen operacions que es poden invocar remotament amb les característiques següents:
  - interfície estable
  - contracte perfectament definit
  - informació de contacte
  - polítiques de seguretat i de nivell de servei
- La granularitat pot ser molt diversa
- Els clients que els invoquen poden a la seva vegada ser serveis
- Els serveis apliquen els dos patrons de sistemes distribuïts vistos:
  - façana remota → controladors de servei
  - DTO → agrupació dels paràmetres
- Freqüentment els serveis s'ofereixen com a serveis Web:
  - ús de les tecnologies conegudes (WSDL, SOAP, UDDI)

Tipus de serveis

Un servei es pot classificar com d'un o més dels tipus següents:

- Petició / resposta
  - obté informació, realitza càlculs i genera un resultat
- Treballador
  - provoca un canvi d'estat en allò sobre què treballa
- Monitor
  - notifica quan es produeix algun canvi rellevant en un estat
- Agent
  - variant de monitor que actua en lloc de només notificar
- Intermediari
  - intercepta una petició de servei de forma transparent, la transforma i envia al destí original
- Agregador
  - combina els resultats de diversos serveis o fonts de dades
- Procés
  - agregador de llarga durada

Combinació de serveis

- Orquestració
  - un servei primari invoca altres serveis
  - coneix la seqüència d'accions, les interfícies dels serveis i les respostes possibles
- Interacció de negoci
  - existeix un mecanisme de coordinació que coneix la informació anterior
  - processos de negoci llargs amb diversos participants
  - característica dels serveis de tipus Procés
- Intercepció
  - un servei intercepta una petició, hi fa algun tractament, i la retorna
  - per a funcions com seguretat, auditoria, traducció de dades, etc.
  - en molts escenaris, tant el servei receptor com l'emisor ignoren l'existència del interceptor
  - característica dels serveis de tipus Interceptor

Característiques (1)

- Alta cohesió
  - implementen un únic concepte de negoci, funció o procés
    - obrir nou compte bancari, autenticar usuari, …
  - poden colaborar amb d'altres serveis per completar la seva feina
- Baix acoblament
  - minimitzen dependències amb el seu context
- Autonomia
  - tenen control sobre la lògica que encapsulen
- Abstracció
  - amaguen la lògica al seu context
  - el contracte és l'únic mitjà de saber què és el que fan

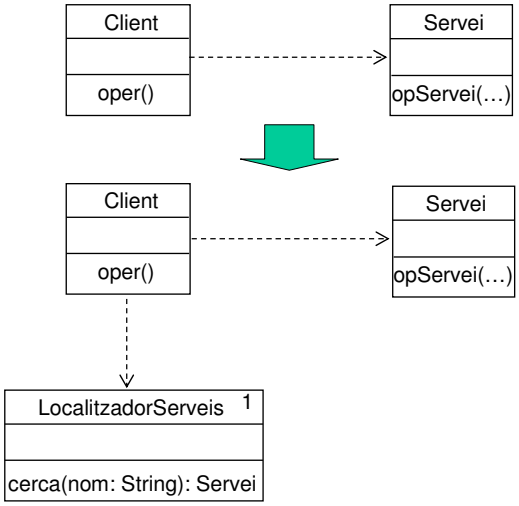
Característiques (2)

- Reusabilitat
  - la lògica és dividida en serveis i operacions amb la intenció de promoure reusabilitat
- Composabilitat
  - col.leccions de serveis poden ser composades per formar nous serveis
- Absència d'estat (*statelessness*)
  - minimitzen la retenció d'informació específica d'una activitat
- *Discoverability*
  - descriptius de manera que puguin ser descoberts i avaluats

Patró Localitzador de Servei

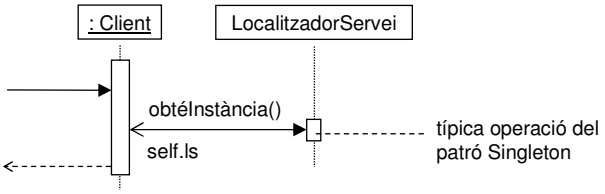
- **Context**
  - En un sistema distribuït és necessari accedir a un servei remot
- **Problema**
  - La localització de serveis implica interfícies complexes i operacions de xarxa
  - El procés pot consumir molts recursos
  - Els clients sempre accedeixen de la mateixa manera
- **Solució**
  - Definim una classe *singleton*, Localitzador Servei, que centralitza el procés de localització
    - En ser *singleton*, es pot crear a l'inici de la sessió o aplicació
  - El client es comunica només amb aquesta classe, usant el nom lògic del servei
  - Localitzador Servei implementa estratègies (e.g., *caching*) per optimitzar recursos i evitar accessos distribuïts redundants

Patró Localitzador de Servei
Aspecte estàtic

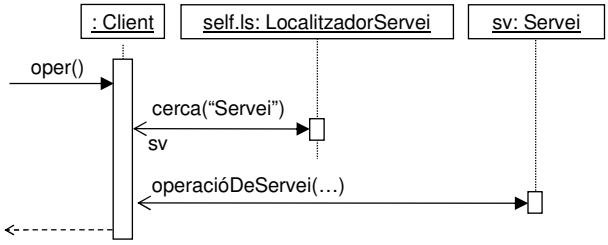


Patró Localitzador de Servei
Aspecte dinàmic

Inicialització



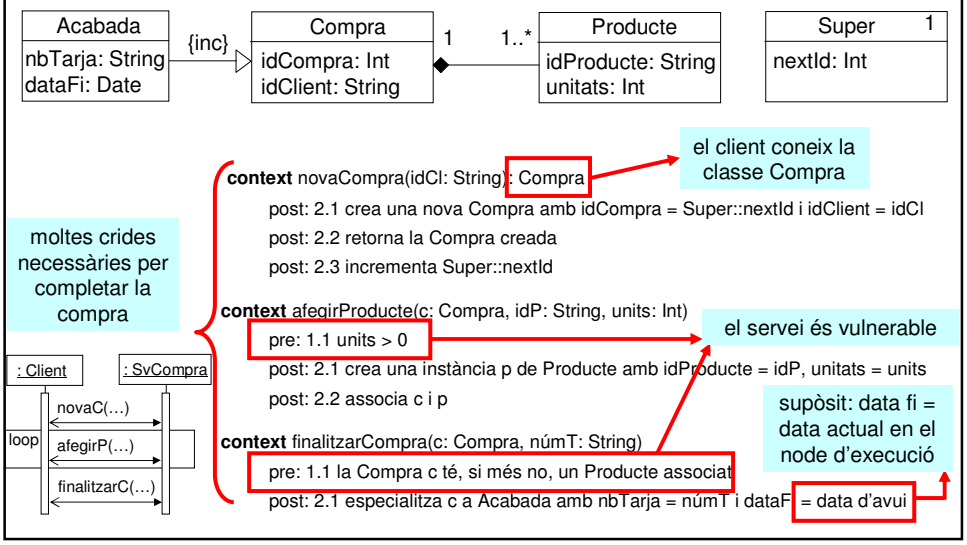
Crida



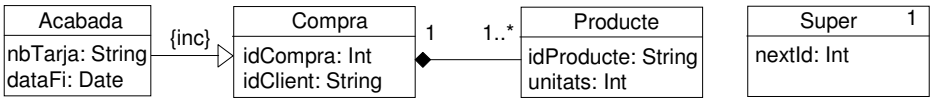
Disseny de serveis

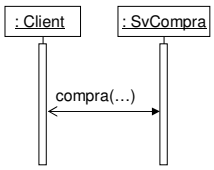
- Dissenyar un servei és similar a dissenyar qualsevol altre tipus d'aplicació
  - cal fer-ne una especificació, un disseny i una implementació
- Cal tenir en compte:
  - distribució: els serveis han de permetre el seu ús eficient en sistemes distribuïts
    - ◊ apliquem els patrons Façana Remota i DTO
  - robustesa: els serveis han d'estar preparats per ser usats en diferents contexts
    - ◊ minimitzar el nombre de precondicions
    - ◊ minimitzar el nombre d'informació "hard-coded"
  - baix acoblament: els serveis no han d'imposar l'ús de classes de domini
    - ◊ les dades enviades a / rebudes dels serveis han de ser tipus simples

Exemple de servei mal dissenyat



Exemple de servei ben dissenyat





context compra(idCl: Int, prods: Set(idP: String + units: Int),
númT: String, dataF: Date)

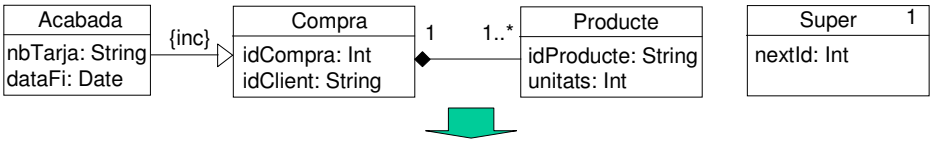
exc 1.1 no-compra-res: prods.size = 0
1.2 mal-producte: existeix algun p dins de prods tal que p.units < 1

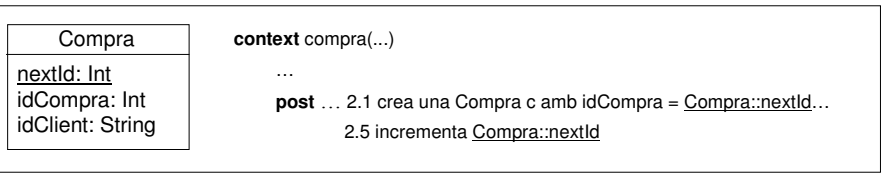
post :
2.1 crea una Compra c amb idCompra = Super::nextId, idClient = idCl
2.2 especialitza la Compra c a Acabada amb nbTarja = númT,
dataFi = dataF
2.3 crea una instància de Producte per cada p dins de prods
2.4 associa cadascuna de les instàncies creades amb la Compra c
2.5 incrementa Super::nextId

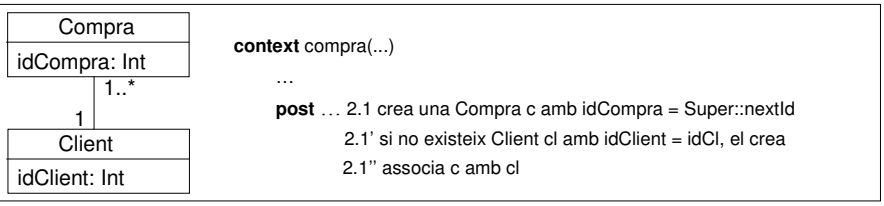
Especificació de serveis: abstracció

- Des de la perspectiva del client d'un servei, l'especificació anterior dóna massa nivell de detall
  - els contractes de les operacions exposen l'estructura del diagrama de classes
  - en realitat, diversos diagrames de classes poden descriure la mateixa situació
  - necessitem descripcions de més alt nivell
- Per això, l'especificació dels serveis:
  - no inclourà el model conceptual de dades
  - conseqüentment, els contractes descriuran els efectes de les operacions sense entrar en detall en els efectes o condicions sobre l'estat del sistema
- De fet, és la situació habitual actualment

Exemple: diversos models conceptuals equivalents





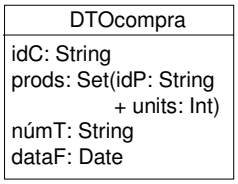


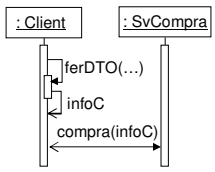
Exemple d'un servei ben dissenyat
Usant contractes abstractes

context compra(idCl: Int, prods: Set(idP: String + units: Int), númT: String, dataF: Date)
exc: 1.1 no-compra-res: prods.size = 0
1.2 mal-producte: existeix algun p dins de prods tal que p.units < 1
post: 2.1 crea una Compra c amb idCompra = Super::nextId, idClient = idCl
2.2 especialitza la Compra c a Acabada amb nbTarja = númT, dataFi = dataF
2.3 crea una instància de Producte per cada p dins de prods
2.4 associa cadascuna de les instàncies creades amb la Compra c
2.5 incrementa Super::nextId

context compra(idCl: Int, prods: Set(idP: String + units: Int), númT: String, dataF: Date)
exc: 1.1 no-compra-res: prods.size = 0
1.2 mal-producte: existeix algun p dins de prods tal que p.units < 1
post: 2.1 enregistra la creació d'una compra feta pel client idCl dels productes inclosos a prods (identificador + unitats), finalitzada la data dataF i pagada amb la tarja de número númT

Contracte final del servei
Usant DTOs amb classes





context compra(infoC: DTOcompra)

exc 1.1 no-compra-res: infoC.prods.size = 0

1.2 mal-producte: existeix algun p dins de infoC.prods tal que p.units < 1

post

2.1 enregistra la creació d'una compra feta pel client infoC.idCl dels productes inclosos a infoC.prods (identificador + unitats), finalitzada la data infoC.dataF i pagada amb la tarja de número infoC.númT

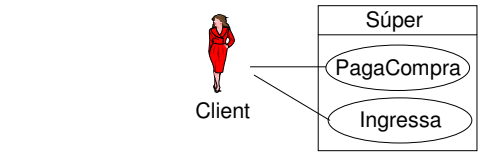
Disseny basat en serveis
Determinant la frontera del sistema

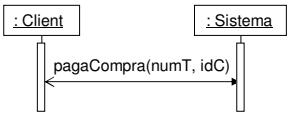
- L'existència de serveis implica conèixer quines dades i funcionalitats són pròpies del SI que dissenyem i quines es consideren externes
  - decisió a prendre en l'especificació: determinar la frontera del sistema
  - aparició d'actors en l'especificació que modelen dades i funcionalitats externes
- El disseny del SI no s'ha de preocupar dels actors software externs a la frontera del sistema

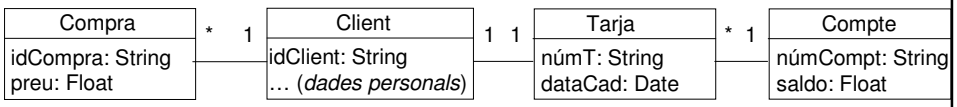
Disseny basat en serveis

Determinant la frontera del sistema: exemple, servei de pagament

Situació: el Súper manega comptes dels seus clients, que paguen les seves compres usant targes vinculades a aquest compte (e.g., El Corte Inglés)







context pagaCompra(númT: String, idC: String)

pre 1.1 existeix la Tarja númT

pre 1.2 dataCad de la Tarja númT ≥ dataAvui()

pre 1.3 existeix la Compra idC i és del client posseïdor de la tarja númT

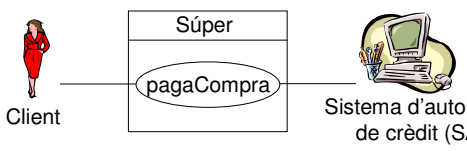
pre 1.4 saldo del Compte de la Tarja númT ≥ preu de la Compra idC

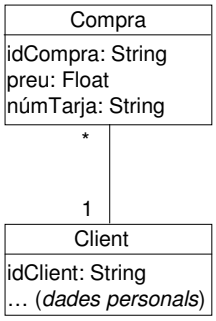
post 2.1 saldo del Compte de la Tarja númT decrementat en preuC

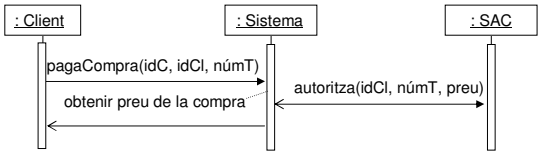
Disseny basat en serveis

Determinant la frontera del sistema: exemple, servei de pagament

Situació: el Súper contracta un servei de pagament extern capaç de processar les targes dels bancs i caixes més habituals







context pagaCompra(idC: String, idCl: String, númT: String)

pre 1.1 existeix la compra idC

pre 1.2 existeix el client idCl

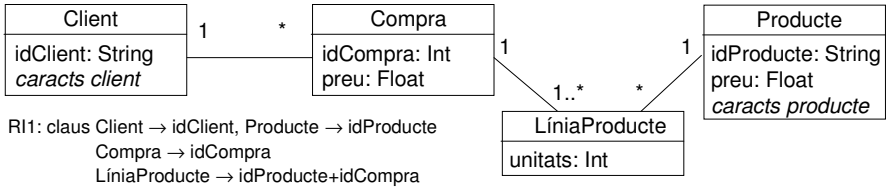
pre 1.3 el sistema d'autorització de crèdit autoritza el pagament del client idCl amb la tarja númT

post 2.1 posa l'atribut númTarja de la compra idC igual a númT

Disseny basat en serveis
Assignació de responsabilitats a serveis

- El procés de disseny d'un SI basat en serveis no és gaire diferent al procés habitual de disseny:
  - simplement, s'assignen algunes responsabilitats a serveis disponibles
- Assignació de responsabilitats a capes →
  assignació de responsabilitats a capes i serveis
  - cal determinar quines parts de l'especificació del sistema queden sota el control dels diferents serveis
  - model conceptual de dades: s'eliminen les classes i associacions manegades pels serveis
    - ◊ conservant la informació mínima necessària (típicament, claus) per poder obtenir les dades que queden sota el control dels serveis
  - contractes: cal transformar o àdhuc eliminar les pre i post sota el control dels serveis
- Client: té la responsabilitat d'orquestrar els diferents serveis:
  - aplicant ARO tant com sigui possible, i vigilant l'acoblament, com sempre, i també l'eficiència, en termes de nombre de crides remotes

Disseny basat en serveis
Assignació de responsabilitats a serveis, exemple: previ assignació responsabilitats



RI1: claus Client → idClient, Producte → idProducte
Compra → idCompra
LíniaProducte → idProducte+idCompra

context compra(idC: Int, idCl: String,
prods: Set(idP: String + units: Int))
pre 1.1 existeix Client cl amb idClient = idCl
pre 1.2 prods.size > 0
pre 1.3 per tot p dins de prods: p.units > 0
exc 1.4 producte-no-existeix: algun producte de prods no existeix
post 2.1 crea una Compra c amb idCompra = idC, preu = suma dels preus dels productes de prods multiplicats per les unitats comprades
2.2 associa c amb cl
2.3 per tot p dins de prods: associa el Producte p amb idProducte = idP amb c, i unitats = p.units

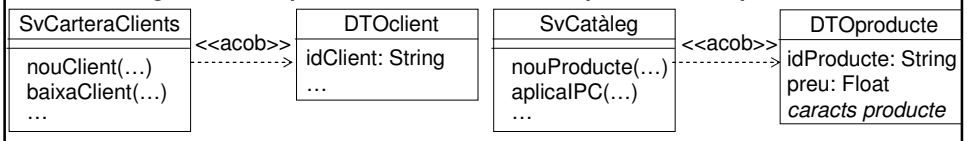
context altaClient(idCl: String, ...)
exc 1.1 client-existeix: existeix el Client idCl
post 2.1 crea un Client cl amb idClient = idCl i les característiques donades

context dadesClient(idC: String): *caracts client*
exc 1.1 client-no-existeix: no existeix el Client idC
post 2.1 retorna les característiques del Client idC

altaProducte, fitxaProducte, pujaIPC, etc.

Disseny basat en serveis

Assignació de responsabilitats a serveis, exemple: serveis disponibles



context SvCarteraClients::nouClient(infoCl: DTOclient)
- **exc** 1.1 client-ja-existeix: ja existeix un client amb identificador infoCl.idClient
- **post** 2.1 crea un client amb identificador infoCl.idClient i les característiques donades

context SvCarteraClients::baixaClient(idCl: Int)
- **exc** 1.1 client-no-existeix: no existeix un client amb identificador idCl
- **post** 2.1 dóna de baixa el client amb identificador idC

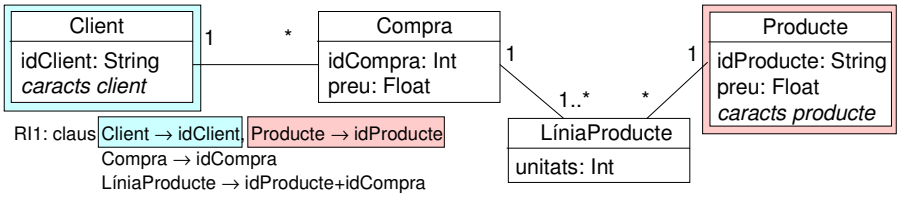
context SvCatàleg::nouProducte(infoP: DTOproducte)
- **exc** 1.1 producte-ja-existeix: existeix un producte amb identificador infoP.idProducte
- **post** 2.1 crea un producte amb identificador infoP.idProducte, preu infoP.preu, i les característiques donades

context SvCatáleg::aplicaIPC(ipc: Float)
- **post** 2.1 incrementa el preu de tots els productes del catàleg amb l'ipc

context SvCatáleg::obtéInfoProducte(idP: String, **out** infoP: DTPproducte)
- **exc** producte-no-existeix: no existeix cap producte amb identificador idP
- **post** 2.1 infoP conté la informació del producte amb identificador idP

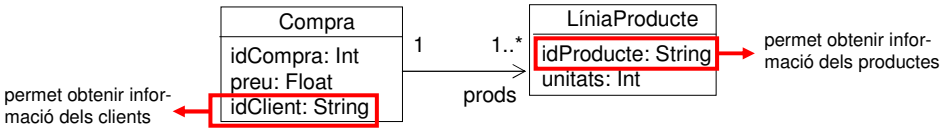
Disseny basat en serveis

Assignació de responsabilitats a serveis, exemple: assignació responsabilitats



RI1: claus Client → idClient, Producte → idProducte
Compra → idCompra
LíniaProducte → idProducte+idCompra

supòsit: navegabilitat →

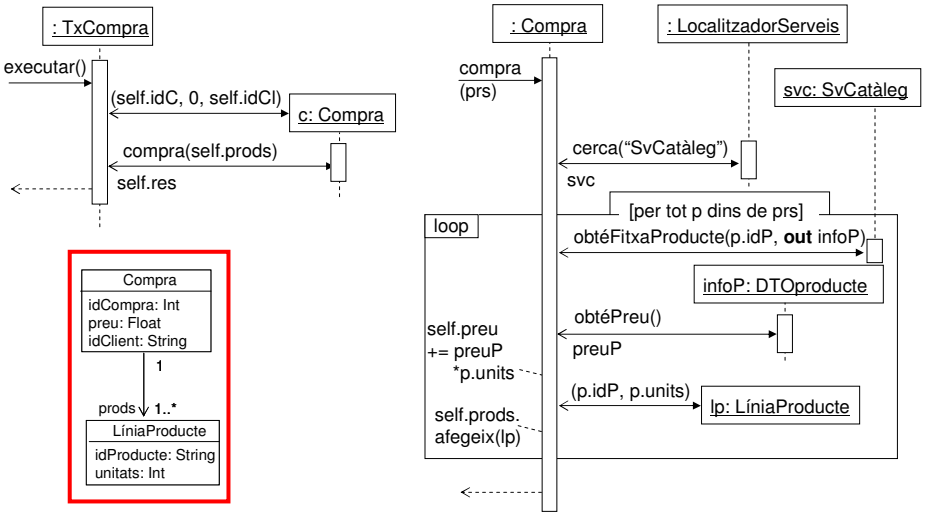


RI1: claus Compra → idCompra, LíniaProducte → idProducte+idCompra

Disseny basat en serveis

Assignació de responsabilitats a serveis, exemple: disseny de les operacions

Cas 1: el catàleg de productes no es pot modificar



Bibliografia

- W. Emmerich
  Engineering Distributed Objects
  Wiley, 2000
- I. Sommerville
  Ingeniería del Software (6a edición)
  Addison-Wesley, 2002, cap. 11
- M. Fowler
  Patterns of Enterprise Application Architecture
  Addison-Wesley, 2003
- E. Roman, S. Ambler, T. Jewell
  Mastering Enterprise JavaBeans
  Wiley, 2002, Segona Edició
- G. Alonso, F. Casati, H. Kuno, V. Machiraju
  Web Services
  Springer Verlag, 2004
- T. Erl
  Service-Oriented Architecture
  Prentice Hall, 2005