

Designing with Services

Software Architecture – Xavier Franch, Cristina Gómez

1

En aquest tema veurem com dissenyar la capa de domini utilitzant serveis externs que proporcionen part de la funcionalitat que el nostres sistema ha de donar. La utilització de serveis en un sistema software no és exclusiu de la capa de domini. Tot i que veurem patrons i tècniques per utilitzar els serveis des de la capa de domini, aquests patrons i tècniques també es poden utilitzar a la resta de capes que utilitzin serveis.

Designing with Services

- Service Identification
- Reassignment of Responsibilities to Services
- Patterns for Accessing Services
- Ensuring Service Availability and Efficiency
- References

Els temes principals que tractarem seran: 1) identificació dels serveis que podem utilitzar en el nostre sistema, 2) assignació de part de les responsabilitats que teníem assignades al nostre sistema (mitjançant la informació representada al diagrama de classes i els contractes de les operacions) als serveis que s'ha identificat, 3) patrons que caldrà utilitzar per invocar als serveis externs que utilitzarem i 4) com assegurarem la disponibilitat dels serveis i la seva eficiència (mesurada com a temps de resposta del sistema).

Service Identification

- It is usual to include previously designed components or services in the design of software systems.
- It makes faster and cheaper to design and build new software systems, since the reused components will not only be already designed but also tested.
- Some of these reused services are identified in the specification phase.
 - For instance, in the NextGen POS system, the third-party tax calculator and inventory control services.

Quan fem l'especificació del sistema, moltes vegades ja coneixem que existeixen components ja desenvolupats que podem utilitzar per tal de no implementar parts del nostre sistema. Si els utilitzem, serà més fàcil i econòmic desenvolupar el nostre sistema. A vegades l'ús de serveis externs és obligatori ja que és inviable que nosaltres els puguem desenvolupar. Per exemple, un servei d'autorització de pagaments. Aquest servei no el podem desenvolupar ja que no tenim la informació necessària per implementar-ho.

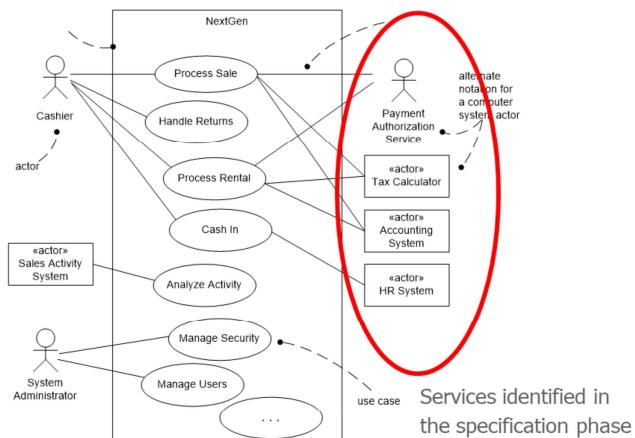
Service Identification

- Other services are identified in the design phase.
- The main reason to identify services in the design phase is to ensure the non-functional requirements (for instance, service availability, efficiency, etc...)
- The way in which our software system will interact with services will depend on the location of the services (local or remote).

A part de la identificació dels serveis que es realitza a la fase d'especificació, la majoria dels serveis s'identifiquen a la fase de disseny per satisfer els requisits no funcionals. Aquesta identificació requereix conèixer els requisits funcionals del sistema que volem dissenyar i tenir coneixement de serveis (i les funcionalitats que proporcionen) que hi ha disponibles, moltes vegades en catàlegs per internet o proporcionats per entitats externes. Per exemple, <http://webservices.gama-system.com/exchangerates.asmx> proporciona un servei amb diferents operacions que proporcionen informació sobre el tipus de canvi i conversió de monedes.

Service Identification

- **Example.** A simplified version of NextGen POS system.

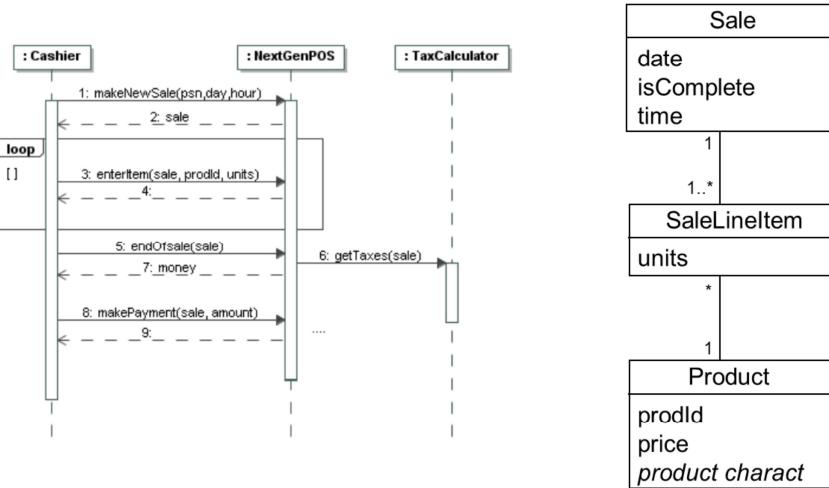


Software Architecture – Xavier Franch, Cristina Gómez

5

A l'exemple que estem utilitzant en aquest tema, 4 serveis externs (emmarcats en vermell) es van identificar a la fase d'especificació i, per tant, es van incloure en el diagrama de casos d'ús del sistema.

Service Identification



RI1: keys Product → prodId
 Sale → date, time
 SaleLineItem → prodId, date, time

Software Architecture – Xavier Franch, Cristina Gómez

6

De la mateixa manera, els diagrames de seqüència d'esdeveniments del sistema que es defineixen a la fase d'especificació també inclouen les interaccions amb els serveis identificats en aquesta fase. Al diagrama de seqüència de la transparència es mostra la interacció amb el servei calculadora de impostos que calcularà a partir d'una venda, el preu total de la mateixa incloent els impostos associats a cada producte comprat. També disposeu a la transparència la part de l'esquema conceptual rellevant pel diagrama de seqüència.

Service Identification

```
context DomainLayer::enterItem(productId: String, units: Integer)
  pre: 1.1 product productId exists
  pre: 1.2 units->0
  post: 2.1 a SalesLineItem instance sli was created
        2.2 sli was associated with the current sale
        2.3 sli.units became units
        2.4 sli was associated with a product with productId

context DomainLayer::endOfSale(): Money
  post: 2.1 the sale became complete
        2.2 result = sale price + taxes (the system calls the getTaxes operation of the
            TaxCalculator service with the current sale as a parameter to obtain the
            taxes)

...
```

A la transparència disposeu dels contractes de dues operacions (a mode d'exemple) del diagrama de seqüència d'esdeveniments del sistema anterior. Els contractes definits a la fase d'especificació inclouen les responsabilitats assignades als serveis identificats en aquesta fase. Per exemple, la postcondició 2.2 del contracte de l'operació endOfSale estableix que el càlcul de les taxes es farà invocant a una operació del serveix del càlcul d'impostos.

Suposeu ara que volem dissenyar les dues operacions de la transparència.

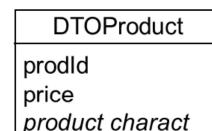
Service Identification

- We identify a service that provides the management of a product catalog. The use of this service may reduce the time and cost of developing our system.
- The operations offered by the service, among others, are the following:

```
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 SvCatàleg::eliminaProducte(idP: String)
  exc 1.1 producte-no-existeix: no existeix cap producte amb identificador idP
  post 2.1 incrementa el preu de tots els productes del catàleg amb l'ipc

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



Software Architecture – Xavier Franch, Cristina Gómez

8

Per dissenyar les operacions anteriors, el primer que hem de fer és veure si existeix algun servei disponible que podríem reutilitzar per evitar dissenyar/implementar tota la funcionalitat de les operacions. Després d'explorar diversos serveis trobem que el servei SVCatàleg proporciona la gestió d'un catàleg de productes. Ens adonem que aquest servei el podem utilitzat per emmagatzemar els productes de la nostra cadena de supermercats i per utilitzar les operacions que proporciona (només es mostren a la transparència algunes operacions del servei). És per això que decidim identificar i utilitzar aquest servei a la fase de disseny per reduir el cost i el temps de disseny i implementació del nostre servei.

Usualment les operacions dels serveis que podeu trobar no tenen uns contractes específics amb excepcions i postcondicions. En general, en comptes de contractes, les operacions dels serveis proporcionen una descripció textual del que fa l'operació. De la mateixa manera, tampoc proporcionen el diagrama de classes intern que utilitza el servei. La informació que manega i enregistra el servei l'hem de deduir nosaltres observant els paràmetres de les operacions i la descripció del que fan les mateixes.

Nosaltres a l'assignatura i per evitar ambigüïtats proporcionarem els contractes de les operacions dels serveis.

Reassignment of Responsibilities to Services

- After the identification of new services in the design phase, some responsibilities of the contracts have to be assigned to the services.
- Classes and associations managed by the services must be removed from the class diagram
 - Keeping the information (basically, keys) to obtain the data managed by the services.
 - Adding classes and operations to the class diagram (as a result of applying design patterns) to use and access services.

Una vegada identificats els serveis (tant en la fase d'especificació com en la fase de disseny) que utilitzarem per dissenyar el nostre sistema, ara cal reassignar part de les responsabilitats que estaven assignades al nostre sistema (mitjançant el diagrama de classes de disseny i el contracte de les operacions) als serveis identificats. Per fer-ho caldrà:

1. Identificar les postcondicions i excepcions dels nostres contractes que no caldrà que nosaltres dissenyem ja que seran garantides pels serveis que utilitzarem. Els nostres diagrames de seqüència de la capa de domini no caldrà que garanteixin aquelles excepcions i postcondicions que garantiran els serveis. Només caldrà invocar (ja veurem més endavant com) des dels nostres diagrames de seqüència a les operacions corresponents dels serveis.
2. Eliminar del nostre diagrama de classes tota aquella informació que enregistren els serveis. Caldrà però mantenir en el nostre sistema aquella informació que ens permeti accedir a la informació que estarà enregistrada en els serveis (bàsicament les claus per obtenir la informació). A més, caldrà afegir al nostre diagrama de classes totes aquelles classes i operacions (usualment derivades de l'aplicació de patrons de disseny) que siguin necessàries per accedir als serveis.

Reassignment of Responsibilities to Services

- **Example:** Some responsibilities of the contracts have to be assigned to the services

```
context DomainLayer::enterItem(productId: String, units: Integer)
  pre: 1.1 product productId exists
  pre: 1.2 units-ok: units>0
  post: 2.1 a SalesLineItem instance sli was created
        2.2 sli was associated with the current sale
        2.3 sli.units became units
        2.4 sli was associated with a product with productId ← Consider the use of SvCatàleg

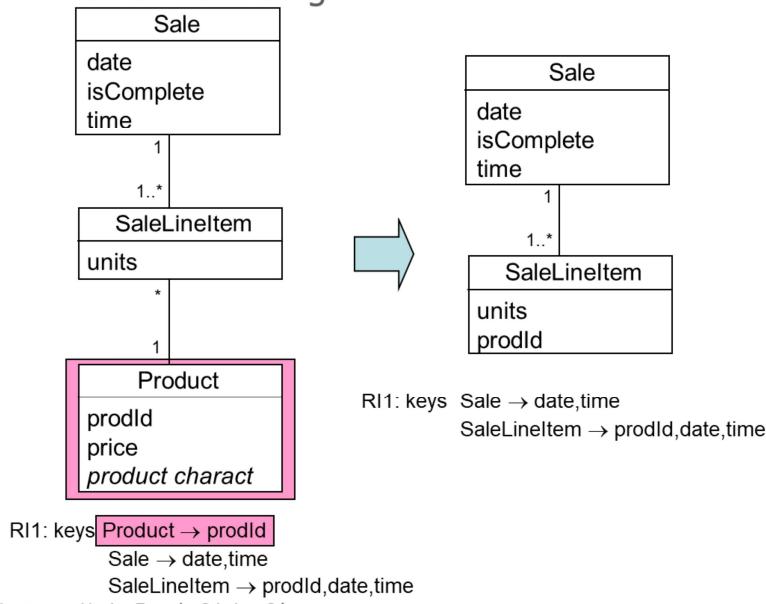
context DomainLayer::endOfSale(): Money
  post: 2.1 the sale became complete
        2.2 result = sale price ← taxes ← Responsibility assigned to TaxCalculator service (all the details in unit 5)
  ...
...
```

Si volem reassignar les responsabilitats dels dos contractes que es mostren a la transparència, primer caldrà identificar les postcondicions i excepcions dels nostres contractes que no caldrà que nosaltres dissenyem ja que seran garantides pels serveis que utilitzarem (pas 1 descrit a la transparència 9).

Per fer-ho, a l'exemple podem veure com hem identificat dues postcondicions (emarcades en vermell) que estaven assignades al nostre sistema (estaven en el contracte de les nostres operacions) i que amb la utilització de dos dels serveis identificats (SvCatàleg i TaxCalculator) caldrà reassignar a aquests serveis.

Reassignment of Responsibilities to Services

- Modification of class diagram



Software Architecture – Xavier Franch, Cristina Gómez

11

Caldrà també eliminar del nostre diagrama de classes tota aquella informació que enregistrin els serveis (part del pas 2 descrit a la transparència 9).

A la part esquerra de la transparència teniu la part del diagrama de classes original que afecta a les operacions que volem dissenyar. Com ara volem utilitzar el SvCatàleg i en aquest servei es poden emmagatzemar els productes de la cadena de supermercats amb el proID, el preu i altres característiques (mireu transparència 8), decidim eliminar del nostre sistema aquesta classe i mantenir exclusivament l'identificador de producte per poder accedir a la resta d'informació del producte que estarà al servei, en el cas de que sigui necessari (com es mostra al diagrama de classes de la dreta). Hi ha una solució alternativa on es podria mantenir la classe Product amb l'identificador (sense la resta d'atributs) i on l'associació entre Product i SaleLineItem hauria de tenir la multiplicitat 1 a la banda de Product i multiplicitat 1..* a la banda de SaleLineItem (noteu que és diferent de l'original). El motiu és que el que representem en aquest diagrama són els productes que alguna vegada s'han venut (per això el mínim de la multiplicitat a la banda de SaleLineItem). En canvi, els productes que s'ofereixen a la venda estan al servei i no els tenim en el nostre sistema.

Discussió important: En aquest cas hem suposat que aquest servei SvCatàleg era un servei dedicat, és a dir, on els productes que hi ha són exclusivament els productes de la nostra cadena de supermercats i que el nostre sistema (o utilitzant altre mecanisme) ha anat afegint aquests productes. Això vol dir que si en algun moment volem saber tots els productes que ven la cadena de

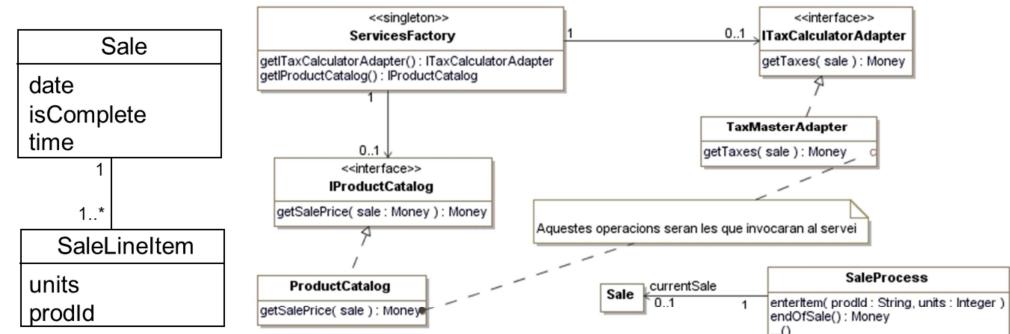
supermercats, ho podríem saber utilitzant una operació del servei que em retorna tots els productes que hi ha al servei. Aquesta operació hauria d'estar definida a la transparència 8 però no apareix ja que no hi són totes les operacions del servei (per manca d'espai a la transparència).

Ara suposeu un escenari diferent. Suposeu que us diuen que el servei SvCatàleg és un servei que emmagatzema productes de diferents cadenes de supermercats. I assumim també, per simplificar, que totes les cadenes venen els seus productes al mateix preu. Això sí, no totes els cadenes venen els mateixos productes. És a dir, la cadena de supermercats per a la qual dissenyem el seu sistema ven un subconjunt dels productes que hi ha al servei. Amb aquestes condicions, quina seria la informació que hauríem d'eliminar (podar) del diagrama de classes de l'esquerra de la transparència? És a dir, quin seria el diagrama de classes del nostre sistema? **Penseu-hi abans de mirar la resposta al paràgraf següent.**

El diagrama de classes del nostre sistema (per a l'escenari descrit) mantindria la classe Product amb només la clau. La resta d'informació estaria al servei. D'aquesta forma es mantindria en el nostre sistema l'identificador de tots els productes que ven la cadena de supermercats ja que l'operació que retorna tots els productes que hi ha definida al servei no em serviria per saber els productes que ven la cadena (em retornaria el conjunt de productes que venen entre totes les cadenes que utilitzen el servei). Si volem obtenir tota la informació dels productes que ven la nostra cadena podem utilitzar l'operació de SVCatàleg obtéInfoProducte que a partir de l'identificador obté tota la informació del producte.

Reassignment of Responsibilities to Services

- Applying design patterns to use services (singleton, factory, adapter)



Software Architecture – Xavier Franch, Cristina Gómez

12

Com a part del pas 2 descrit a la transparència 9, també caldrà afegir al nostre diagrama de classes totes aquelles classes i operacions (usualment derivades de l'aplicació de patrons de disseny) que siguin necessàries per accedir als serveis. Per accedir als serveis serà necessari utilitzar els patrons següents:

- Adaptador: aquest patró l'utilitzarem per evitar l'acoblament del nostre sistema amb el servei que invoquem. Minimitzar aquest acoblament permetrà en molts casos canviar el servei i no haver de modificar el nostre sistema. Definirem en els adaptadors (interface) les operacions que el nostre sistema necessita i definirem el mètode d'aquestes operacions a la classe que implementa la interface. Aquests mètodes hauran de fer les invocacions al/s serveis. Reviseu el patró adaptador (Unit 3.2.2), si teniu dubtes.
- Factoria: per poder obtenir les referències dels adaptadors i poder invokes a les operacions d'aquests adaptadors usarem el patró factoria. Això ho farem de forma similar a com ho feiem amb els controladors d'accés a les dades (que recordieu que també eren adaptadors).
- Singleton: els factories són singletons.

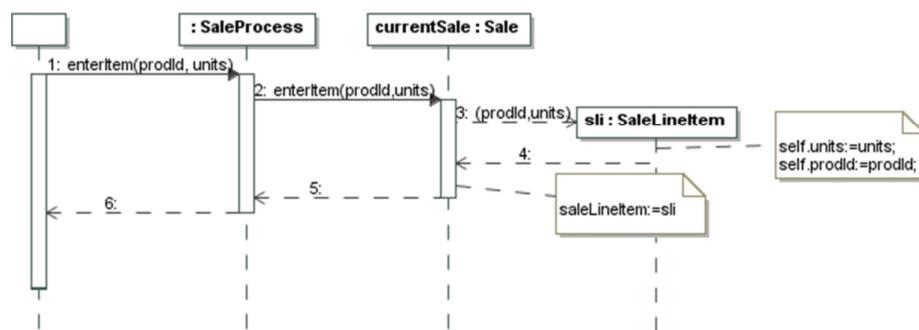
A la transparència teniu les classes que s'han afegit com a conseqüència de l'aplicació d'aquests patrons a l'exemple que estem utilitzant.

També podeu observar a la transparència que hem utilitzat una classe SaleProcess on hi ha definides les dues operacions que estem dissenyant.

Aquesta classe és un controlador (però no de tipus transacció com els que hem utilitzat fins ara) sinó de tipus cas d'ús. Aquests controladors s'utilitzen quan volem definir casos d'ús que tenen un conjunt d'esdeveniments (com és el cas de l'exemple) i entre aquests esdeveniments/operacions es vol compartir informació. A l'exemple la informació que s'està compartint és la venda actual. El primer esdeveniment del cas d'ús la crea i la resta d'esdeveniments l'utilitzen.

Reassignment of Responsibilities to Services

- Sequence diagrams



Assuming navigability: Sale -> SaleLineItem and SaleLineItem -> Product

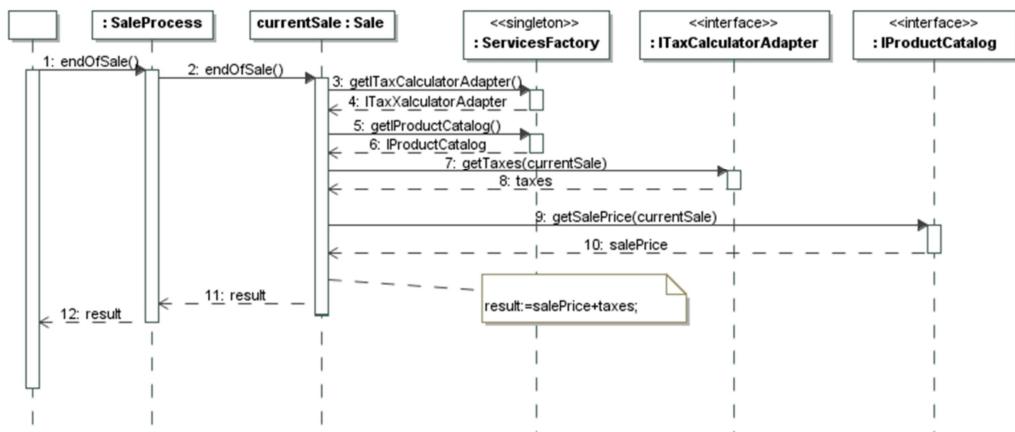
Software Architecture – Xavier Franch, Cristina Gómez

13

A la transparència disposeu del diagrama de seqüència de l'operació enterItem que està definida al controlador de cas d'ús. Aquesta operació utilitzava la venda actual per invocar a l'operació amb el mateix nom de Venda. Aquesta operació crea una línia de venda i li assigna les unitats i el identificador de producte i forma la navegabilitat des de venda a línia de venda. Noteu que aquest diagrama de seqüència utilitza el diagrama de classes de la transparència 11 (part dreta).

Reassignment of Responsibilities to Services

- Sequence diagrams



It remains to define the sequence diagrams for accessing services

A la transparència disposeu del diagrama de seqüència de l'operació endOfSale que està definida al controlador de cas d'ús. Aquesta operació utilitz a la venda actual per invocar a l'operació amb el mateix nom de Venda. Aquesta operació demana al factoria les dues referències dels adaptadors i les utilitza per invocar les operacions que hem definit als adaptadors (nosaltres som els responsables de definir aquestes operacions als adaptadors com a part del nostre disseny). Amb l'obtenció del preu de la venda i dels impostos a aplicar, fem la suma per emmagatzemar el resultat en una variable que retornarem. Noteu que aquest diagrama de seqüència utilitza el diagrama de classes de la transparència 11 (part dreta).

Patterns for Accessing Services

- To access services we must take into account:
 - How to localize services
 - Coupling between the classes of our system and services
 - How to communicate data between our system and services
- Some patterns can be applied to access services
 - Service Locator Pattern
 - Data Transfer Object Pattern

A part dels patrons que utilitzem quan fem la reassignació de responsabilitats a serveis (adaptador, factoria i singleton), quan accedim als serveis cal considerar i aplicar altres patrons addicionals:

- Com localitzem els serveis. Per localitzar-los aplicarem el patró Service Locator.
- Com minimitzem l'acoblamet entre les classes del nostre sistema i els serveis. Això ho aconseguim aplicant els patrons adaptador, factoria i singleton.
- Com fem la comunicació entre les dades que tenim al nostre sistema i les dades que utilitza el servei. Per fer-ho aplicarem el patró DTO (Data Tranfer Object).

A continuació explicarem els dos nous patrons.

Patterns for Accessing Services: Service Locator

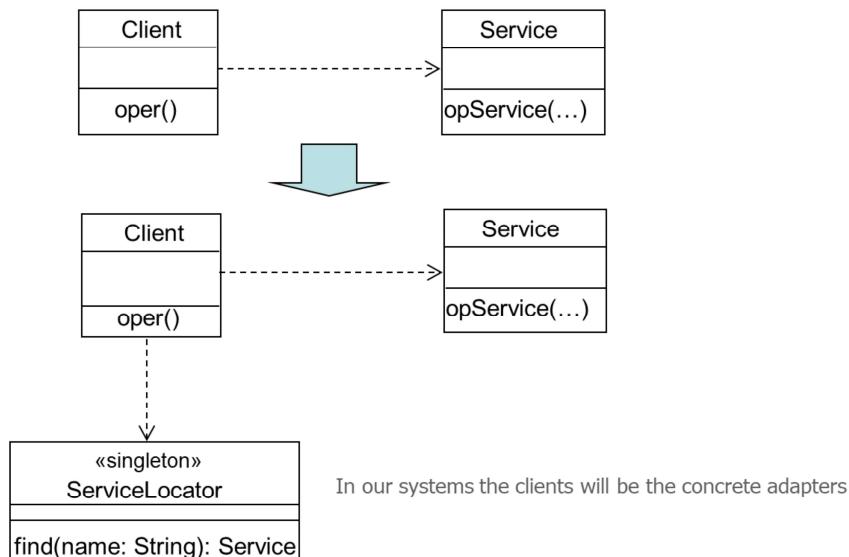
- Context
 - Systems that have to access remote services
- Problem
 - The service localization implies complex interfaces and network operations
 - The process can consume many resources
 - The clients always access the same way
- Solution
 - We define a singleton *class*, Service Locator, that centralizes the localization process
 - Being a singleton, it can be created when initiating the session or application
 - The client only communicates with this class, using the service's logic name
 - Service Locator implements strategies (e.g., caching) to optimize resources and avoid redundant distributed accesses

Software Architecture – Xavier Franch, Cristina Gómez

16

La invocació a les operacions dels serveis implica localitzar els serveis, és a dir, obtenir l'adreça física dels serveis (referències) i la utilització d'operacions de xarxa que res tenen a veure amb la lògica del sistema que s'està desenvolupant. Per poder amagar aquesta adreça física i la forma d'accendir-ne, i per tal d'evitar que aquest procés s'hagi de repetir cada vegada que volem accedir a un determinat servei, el patró Service Locator proposa definir una classe singleton, anomenada Service Locator, que tindrà una operació per obtenir a partir del nom del servei, la seva referència per poder ser invocat. Aquesta operació amagàrà la lògica complexa per obtenir aquesta referència.

Patterns for Accessing Services: Service Locator



Software Architecture – Xavier Franch, Cristina Gómez

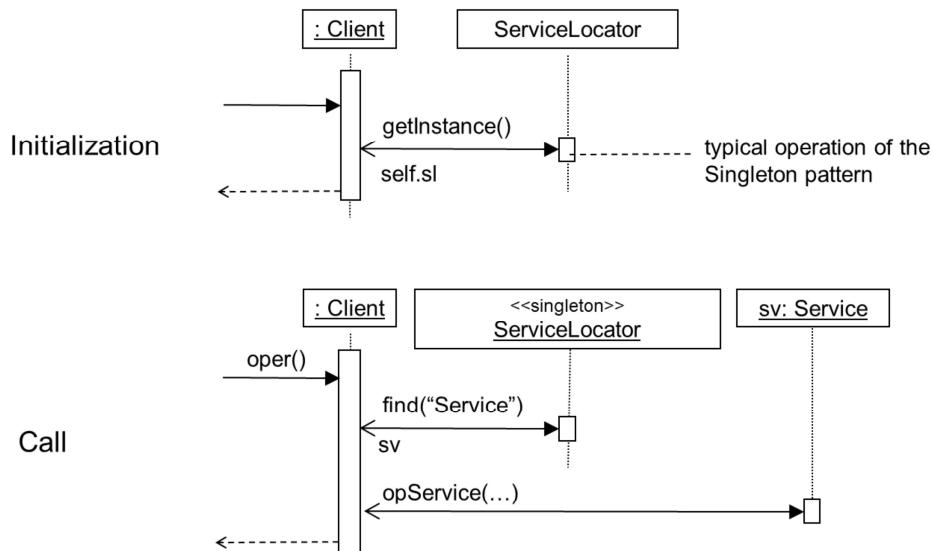
17

A la part superior de la transparència (sense l'aplicació del patró ServiceLocator) es mostra com cada client que volgués invocar a un servei hauria de tenir a les seves operacions la lògica necessària per obtenir l'adreça física. Aquesta lògica estaria repetida (codi repetit) a cada operació que invoqués al servei. Aquesta solució té problemes de manteniment degut a l'existència de codi repetit.

La solució des del punt estàtic que proposa el patró Service Locator consisteix en definir una classe singleton ServiceLocator amb una operació find que a partir del nom del servei obté la referència del servei per poder ser invocat.

Així, quan les operacions d'una classe client (en el nostre cas seran els adaptadors) vulguin invocar a un servei, primer demanaran al ServiceLocator la seva referència a partir del seu nom. Amb aquesta referència les operacions podran invocar a les operacions del servei. La utilització del ServiceLocator fa que si l'adreça física del servei canvia (per exemple, es mou el servei a una altra màquina), la classe client no s'assabenta d'aquest fet, ja que és el ServiceLocator qui amaga aquesta referència.

Patterns for Accessing Services: Service Locator



Software Architecture – Xavier Franch, Cristina Gómez

18

La solució des del punt de vista dinàmic del patró Service Locator consistirà en obtenir l'adreça del servei en els diagrames de seqüència mitjançant l'operació `find`, abans d'invocar a les operacions del servei. Recordeu que nosaltres utilitzem la notació compacta (2on diagrama de seqüència) per invocar a les operacions d'una classe singleton.

Patterns for Accessing Services: Data Transfer Object

- Context
 - Systems need to communicate data to services in an efficient manner
- Problem
 - To minimize the number of calls in the system, each call has to carry more information in the parameters and in the return value
 - The resulting list of data may be long
- Solution
 - We define (use) a new data group that contains all the data that has to be passed as parameter or result. The objects that are these kind of groups are called Data Transfer Objects (DTO)
 - This grouping can be defined as a class (DTO class) with getter and setter operations, or else as a tuple, if the underlying language supports this construction

Software Architecture – Xavier Franch, Cristina Gómez

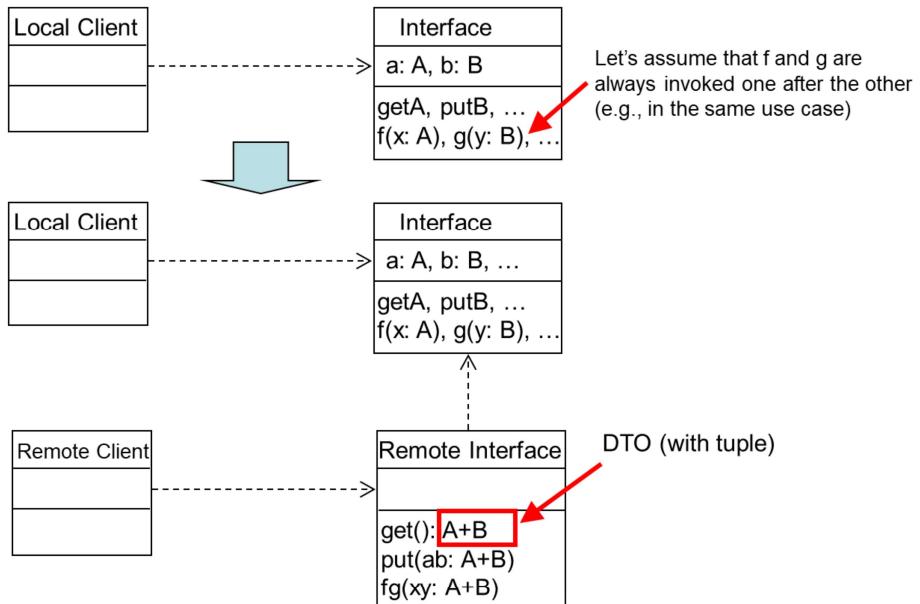
19

Un altre aspecte que hem de considerar quan invoquem als serveis és com es farà la comunicació de dades entre el nostre sistema i els serveis. Aquesta comunicació pot generar dos potencials problemes:

- Les invocacions remotes són més ineficients (respecte al temps de resposta) que les locals. Si en fem moltes invocacions, el temps de resposta s'incrementarà molt. Per evitar això, s'haurien de fer menys invocacions però que en cada una d'elles hi circulessin més dades.
- Quan es fa una invocació remota, les dades que circulen per la xarxa de forma disagregada (molts paràmetres) ho fan de forma més ineficient que si circulen de forma agrupada.

El patró DTO evita aquests problemes proposant que les dades circulin entre el sistema i els serveis de forma agrupada en DTOs (grups de dades).

Patterns for Accessing Services: Data Transfer Object



Software Architecture – Xavier Franch, Cristina Gómez

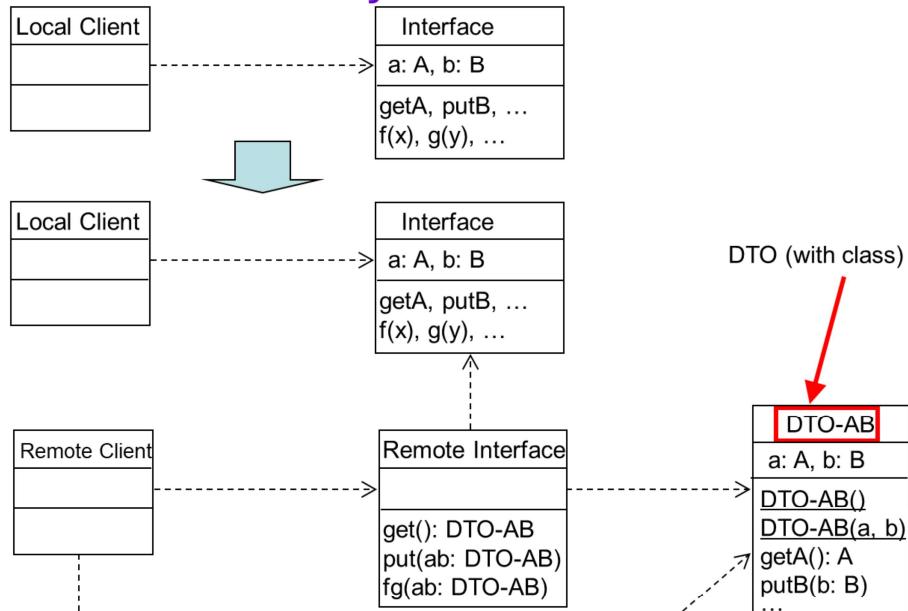
20

A la part superior de la transparència (sense l'aplicació del patró DTO) es mostra com un client que volgués invocar a determinades operacions de forma local podria invocar diverses operacions i en cada una d'elles passar els paràmetres que de forma disagregada. En canvi, si el client volgués invocar a les mateixes operacions de forma remota, aquestes invocacions serien molt costoses (invocacions de diferents operacions f i g, i paràmetres disagregats A i B).

La solució des del punt estàtic que proposa el patró DTO consisteix en definir grups de dades que siguin enviades com a paràmetre per tal de fer la comunicació entre el client i el servei.

Pel cas de l'exemple, en comptes de tenir l'operació f i g amb els paràmetres A i B, podem definir una única operació fg que tingui com a paràmetre A+B (agrupació en tupla).

Patterns for Accessing Services: Data Transfer Object



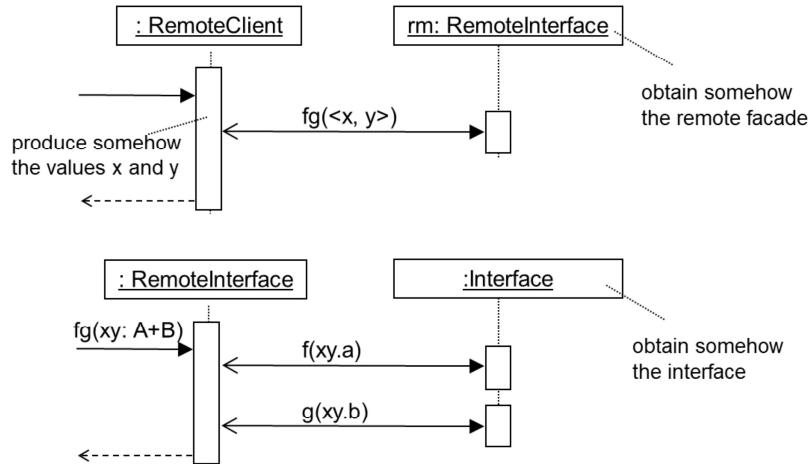
Software Architecture – Xavier Franch, Cristina Gómez

21

A la transparència anterior hem vist l'agrupació de dades en tupla. Hi ha llenguatges de programació que no permeten la definició de tuples. També es poden agrupar les dades mitjançant la definició de classes d'objectes. Per exemple, en el cas de l'exemple (i com a alternativa a l'ús de les tuples) es pot definir la classe DTO-AB i definir en aquesta classe dos atributs, A i B.

Patterns for Accessing Services: Data Transfer Object

- DTO with Tuple



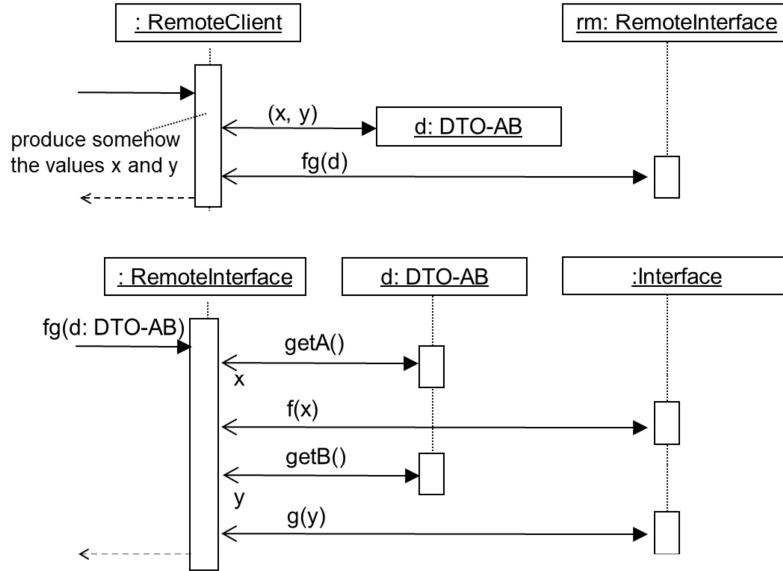
Software Architecture – Xavier Franch, Cristina Gómez

22

La solució des del punt de vista dinàmic del patró DTO és manegar les dades que circulen de forma agrupada en les operacions. Per exemple, en el primer diagrama de seqüència podeu observar com es fa una invocació de l'operació `fg` on el paràmetre és una tupla. La classe que rep la invocació `RemotelInterface` obté cada dada individual i les utilitza per invocar de forma local a les operacions `f` i `g` (segon diagrama de seqüència).

Patterns for Accessing Services: Data Transfer Object

- DTO with Class



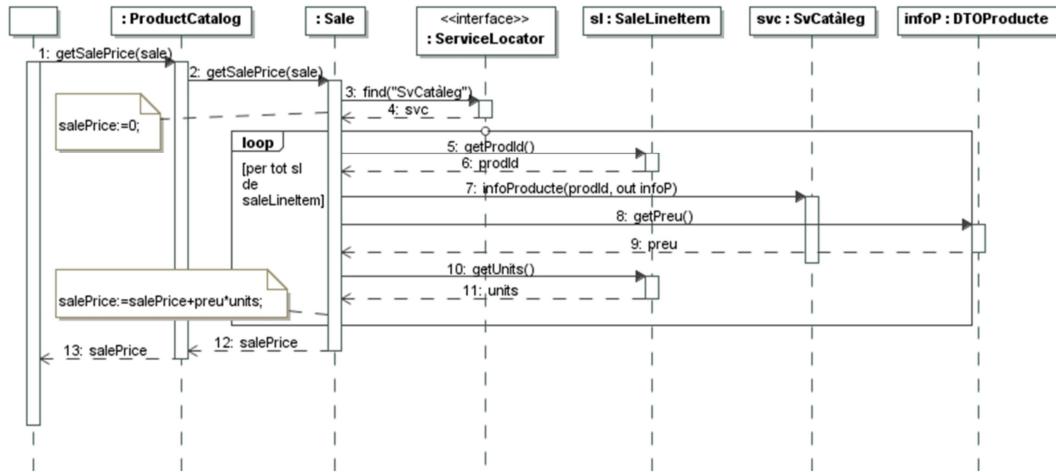
Software Architecture – Xavier Franch, Cristina Gómez

23

La solució des del punt de vista dinàmic del patró DTO és manegar les dades que circulen de forma agrupada en les operacions. Per exemple, en el primer diagrama de seqüència podeu observar com es fa una invocació de l'operació `fg` on el paràmetre és un objecte. La classe que rep la invocació `RemoteInterface` obté el valor de cada atribut individual i els utilitza per invocar de forma local a les operacions `f` i `g` (segon diagrama de seqüència).

Patterns for Accessing Services: Service Locator and Data Transfer Object

- Example



Software Architecture – Xavier Franch, Cristina Gómez

24

A la transparència disposeu del diagrama de seqüència de l'operació `getSalePrice` de la classe `ProductCatalog` (que implementa la interface `IProductCatalog`) i que forma part de l'adaptador cap al `SvCatàleg`. Aquesta operació s'havia invocat al diagrama de seqüència de la transparència 14. En aquest diagrama de seqüència podeu observar l'ús del patró `ServiceLocator` i del patró `DTO`.

Ensuring Service Availability and Efficiency

- Design patterns and principles may be applied to satisfy non-functional requirements
- Some of the design patterns studied until now help to ensure some non-functional requirements. For instance, adapter and service locator patterns provide maintenance.
- Now, we focus on efficiency and service availability non-functional requirements

Fins ara els patrons de disseny que hem vist per invocar els serveis ens permeten millorar la canviabilitat (adaptador, service locator) i el temps de resposta (DTO).

Ara ens centrem en dos requisits no funcionals que usualment s'especifiquen quan s'utilitzen serveis externs: l'eficiència en el temps de resposta i la disponibilitat del serveis. Anem a veure com podem assegurar aquests dos requisits no funcionals en els nostres dissenys.

Ensuring Service Efficiency

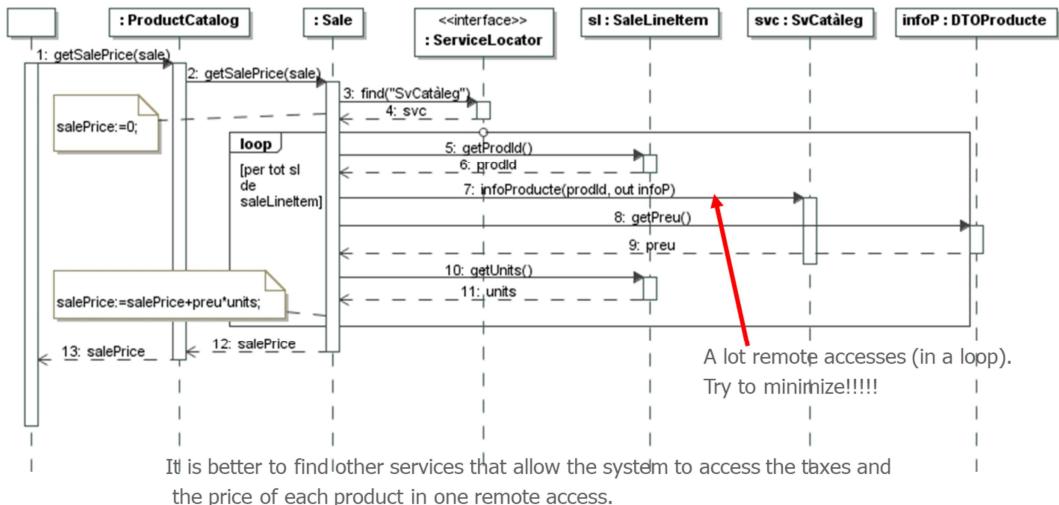
- Efficiency may be achieved in different ways.
 - Applying the adapter and factory pattern to design a cache mechanism to avoid multiple accesses to a remote service.
 - We ensure service efficiency minimizing the remote accesses to services in the following way:
 - We use services that minimize remote accesses and discard the ones that generate multiple remote accesses
 - We use services that allows us to pass the data in a grouped manner (use of the DTO pattern)

Per assegurar l'eficiència (mesurada en un temps de resposta baix) ho podem fer de diferents formes:

- Dissenyar un mecanisme de cache per evitar accedir múltiples vegades al servei. Cada invocació al servei suposa un accés remot amb un temps de resposta que s'incrementa. Si amb un mecanisme de cache aconseguim reduir el nombre d'invocacions, estarem baixant el temps de resposta del nostre sistema. Un exemple de com dissenyar un mecanisme d'aquest tipus el tindreu disponible a l'apartat c) del problema 4.
- Seleccionar les operacions dels serveis que invoquem per tal de minimitzar el temps de resposta. Això vol dir descartar aquelles operacions dels serveis que impliquin fer moltes invocacions, per exemple, dins d'un loop. Utilitzar també aquelles operacions dels serveis on les dades passin agrupades i que permetin que en una única invocació podem obtenir un conjunt gran de dades (utilitzant el DTO).

Ensuring Service Efficiency

- Review of the getSalePrice operation of ProductCatalog



Software Architecture – Xavier Franch, Cristina Gómez

27

Si revisem el diagrama de seqüència de l'operació getSalePrice de la transparència 24 podem observar que no estem assegurant el requisit no funcional d'eficiència. Fem un accés al servei dins del loop. Això implica múltiples accessos remots i per tant un temps de resposta alt. Per solucionar aquest problema caldria buscar un altre servei que permeti obtenir la informació d'un conjunt de productes en un únic accés. Per exemple, amb una operació infoProductes(prods:Set(prodId:String), out infoProds:Set(infoP:DTOProduct)).

Ensuring Service Availability

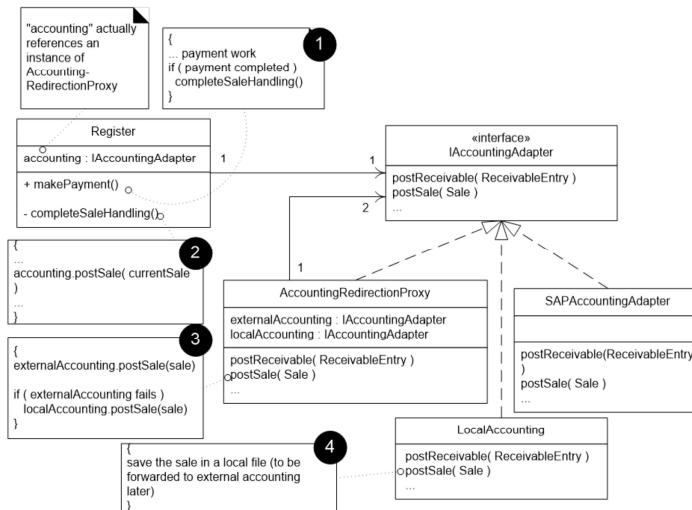
- The systems that we design require a high service availability.
- There are two options to ensure service availability:
 - Using services with a service level agreement that impose a high availability of the service.
 - Design a solution that ensures a high availability of the service.
- There are lots of ways to design solutions with high service availability. For instance, one option could be ensuring service availability using the proxy pattern in the following way:
 - The proxy registers the service and its backup.
 - If the service fails the proxy redirects the request to the backup service

Per assegurar una disponibilitat alta del servei ho podem fer de diferents formes:

- Utilitzar serveis que assegurin una alta disponibilitat. La majoria dels serveis estableixen el nivell de disponibilitat que tenen en forma de percentatge. Per exemple, si el percentatge de disponibilitat és del 80% vol dir que el 80% del temps el servei estarà disponible. El 20% restant el servei no estarà disponible (per exemple, està caigut per algun motiu, està en manteniment, etc...). Una forma d'assegurar una disponibilitat alta és utilitzar serveis que estableixen una disponibilitat alta.
- Dissenyar una solució per garantir l'alta disponibilitat dels serveis. Normalment optem per aquesta solució quan el nivell de disponibilitat ha de ser molt alt i els serveis que utilitzem no ens ho proporcionen. Una possible solució de disseny és tenir un servei “principal” que proporcioni un conjunt d’operacions i un o més serveis de backups per ser utilitzats en el cas de que el servei “principal” no estigui disponible. D'aquesta forma augmentarem el nivell de disponibilitat dels serveis que utilitzem. Per dissenyar aquesta operació podem utilitzar el patró Proxy o Representant que s’encarrega de redireccionar la invocació a un servei de backup si el servei principal no està disponible. A la transparència següent tenir el diagrama de classes on s’ha aplicat una solució d'aquest tipus.

Ensuring Service Availability

- Example



Software Architecture – Xavier Franch, Cristina Gómez

29

Suposeu que tenim una operació makePayment a la classe Register que després de realitzar el pagament, l'ha de registrar en un sistema de comptabilitat que és accessible via un adaptador a un servei (IAccountingAdapter). Aquest sistema de comptabilitat ha de tenir una disponibilitat alta per poder registrar tots els pagaments que es fan. Desafortunadament, el servei on registrem aquest pagaments (accessible via la classe SAPAccountingAdapter) no té la disponibilitat desitjada. És per això, que definim un servei de backup que permet registrar els pagaments en un fitxer local. Aquest servei serà accedit des de l'adaptador (en concret, des de la classe LocalAccounting). Addicionalment, i per gestionar que quan el servei principal no doni resposta, es redireccioni la invocació al servei de backup, definim la classe AccountingRedirectionProxy que actua com a representant. Aquesta classe que té les mateixes operacions que les altres dues i te l'apuntar a les altres dues classes que implementen la interface, invoca primer al adaptador del servei principal i si aquest no respon redirecciona la invocació a l'adaptador del backup. El diagrama de classes de disseny que assegura una alta disponibilitat del servei és el que es mostra a la transparència. Noteu que si voleu incrementar encara més la disponibilitat del servei podríem tenir un segon servei de backup i afegir el seu adaptador.

References

- *Applying UML and Patterns*
C. Larman
Prentice Hall, 2005 (Third edition)
- *Service-Oriented Architecture (SOA): Concepts, Technology, and Design*
T. Erl
Pearson Education, 2005