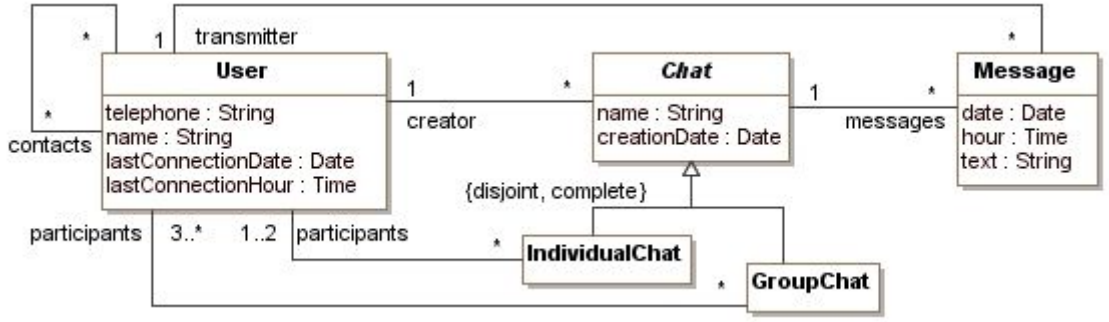
Anàlisi i Disseny amb Patrons

PAC 2: Patrons de Disseny i Assignació de Responsabilitats

Suposem que estem desenvolupant un programari de missatgeria electrònica i que disposem del següent diagrama de classes:



Com podem veure, un usuari té un telèfon que l'identifica, un nom, data i hora de l'última connexió. A més, un usuari té altres usuaris com a contactes i és el creador d'un conjunt de xats. Cada xat té un nom i una data de creació i té un conjunt de missatges. Un xat s'identifica pel seu nom i el telèfon del seu creador. Els xats poden ser individuals o grupals. Els individuals tenen com a molt dos participants i els grupals tenen com a mínim 3 participants. El creador del xat ha de ser un dels seus participants. Els altres participants del xat han de ser contactes de l'usuari creador. Els missatges d'un xat disposen d'una data, hora, text i tenen un emissor. Un missatge s'identifica per l'identificador del xat (nom del xat i telèfon del seu creador) i la data i hora del missatge. Els emissors dels missatges han de ser participants del xat. La data d'un missatge ha de ser posterior a la data de creació del xat.

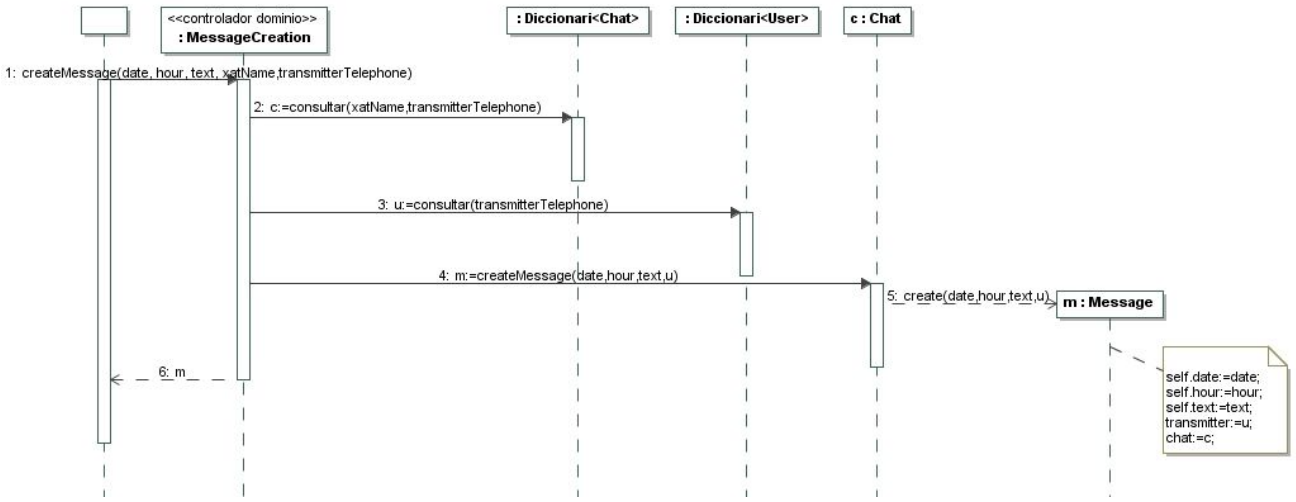
Pregunta 1 (25%)

Suposem que el nostre sistema ha de ser capaç de crear missatges als diferents xats. Per crear aquests missatges és necessari disposar de la informació de la data, hora, text i emissor del missatge, així com del xat on s'ha de publicar el missatge. Es demana:

a) Quins patrons de disseny utilitzaries per aquest cas?
b) Explica quina serà la classe responsable de crear els missatges i justifica la teva resposta (poden haver-hi diferents classes responsables però només heu de proporcionar una).
c) Mostra el diagrama de seqüència o el pseudocodi de l'operació que crea un missatge, inicialitza la seva data, hora i text i l'associa el seu emissor i el seu xat. Useu els diccionaris (mòdul 3, seccions 5.2.2 i 5.2.3) per recuperar instàncies a partir del seu identificador.

Solució

a) Utilitzarem el patró *Creador* ja que ens cal crear instàncies de la classe *Message*.
b) Assignem a la classe *Chat* la responsabilitat de crear un missatge ja que aquesta classe enregistra els objectes missatge. Podríem també assignar aquesta responsabilitat a la classe *User* ja que també enregistra els objectes missatge o al controlador *MessageCreation* ja que té les dades d'inicialització que es passaran al missatge quan aquest sigui creat.
c) El diagrama de seqüència mostra com el controlador obté dels diccionaris els objectes xat i usuari. L'objecte xat és utilitzat per invocar a l'operació *createMessage* i l'objecte usuari es passa com a paràmetre a aquesta operació. Des de l'operació *createMessage* s'invoca a la constructora de Message. Aquesta constructora inicialitza els atributs de l'objecte creat així com l'associació amb l'usuari i amb el xat.



Pregunta 2 (25%)

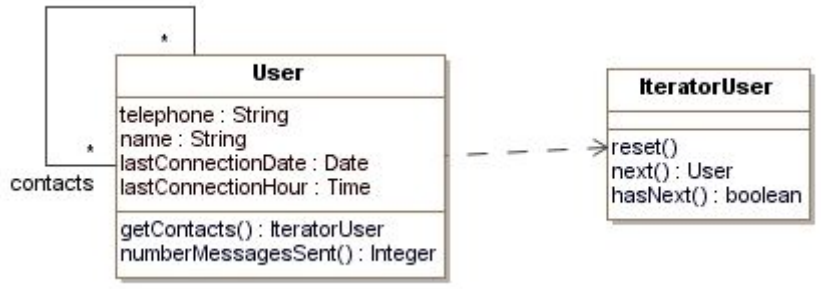
Suposem ara que volem definir una operació *mostActiveContact(): User* de la classe *User* que retorna el contacte de l'usuari *self* (sobre el que s'invoca l'operació) que ha estat emissor de més missatges. Es demana:

a) Quins patrons de disseny utilitzaries per dissenyar aquesta operació? Justifica la teva resposta.
b) Mostra la part del diagrama de classes que es veu modificada com a resultat de l'aplicació dels patrons utilitzats.
c) Mostra el diagrama de seqüència o el pseudocodi de l'operació *mostActiveContact(): User* de la classe User.

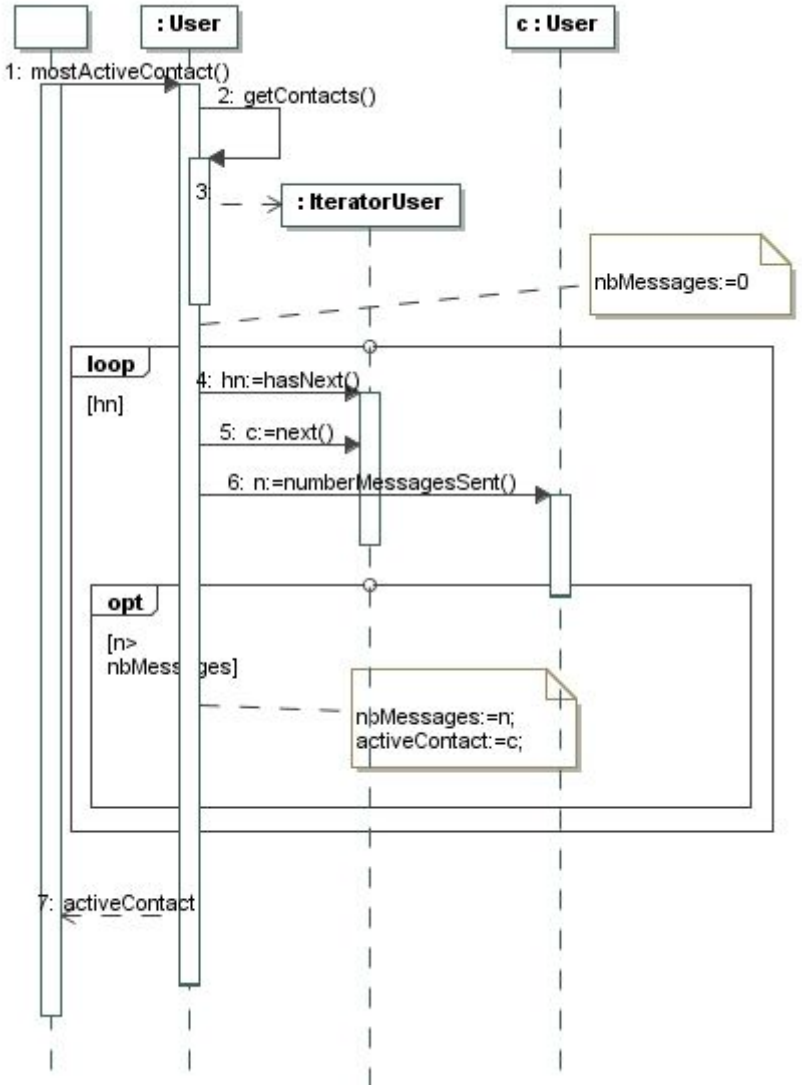
Solució

a) Utilitzarem el patró *Iterador* que permeti recórrer els contactes d'un usuari encapsulant l'estructura interna d'aquesta informació.

b) El diagrama mostra la classe User i l'iterador dels usuaris per recórrer els contactes de l'usuari.



c) El diagrama de seqüència mostra el recorregut dels contactes de l'usuari mitjançant l'iterador. Per cada contacte obtingut en el recorregut es demana el nombre de missatges enviats i es troba el contacte que té més missatges enviats.



Pregunta 3 (25%)

Volem afegir dues funcionalitats al programari que estem desenvolupant. Aquestes funcionalitats han de permetre als usuaris activar i inactivar els seus xats. La funcionalitat que inactiva un xat enregistra la data d'inactivació i no tindrà cap efecte si el xat està inactiu. La funcionalitat que activa un xat enregistra la data d'activació i no tindrà cap efecte si el xat està actiu. Quan es crea un xat

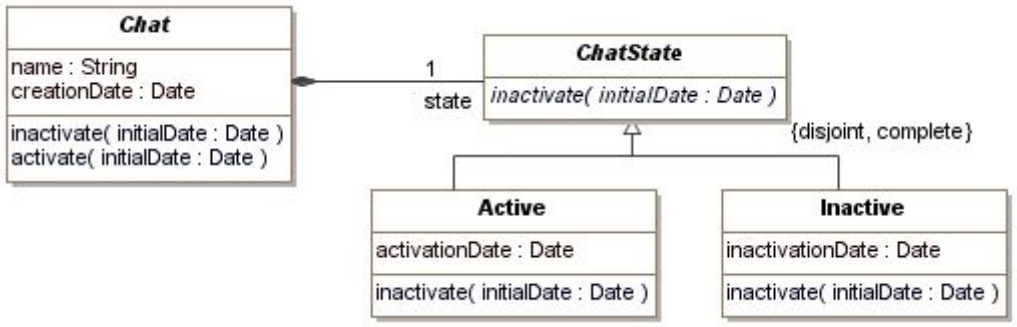
es crea en estat actiu amb la data d'activació igual que la data de creació. El comportament d'un xat inactiu és diferent del comportament d'un xat actiu. Per exemple, un xat inactiu no pot enregistrar missatges.

En concret, volem afegir una operació a la classe *Chat* anomenada *inactivate(initialDate: Date)* que implementi la funcionalitat d'inactivació esmentada. No cal que implementeu la funcionalitat d'activació.

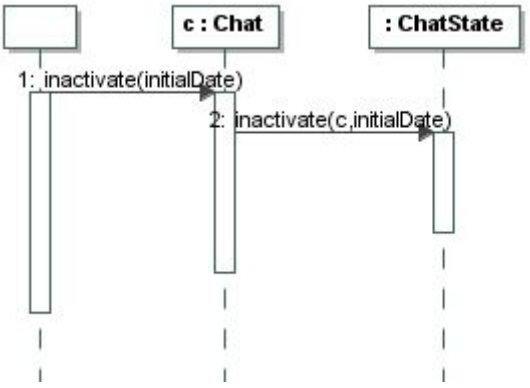
a) Quin patró de disseny recomanaries per aquest cas (si és que en recomanes algún)?
b) Mostra el diagrama de classes resultant d'afegir aquestes funcionalitats.
c) Mostra el diagrama de seqüència o el pseudocodi de l'operació *inactivate* de la classe *Chat.*

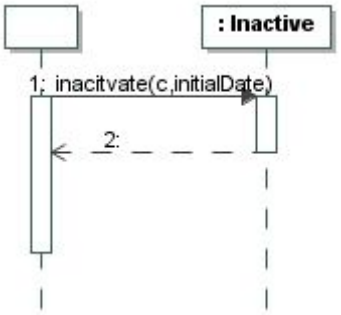
Solució

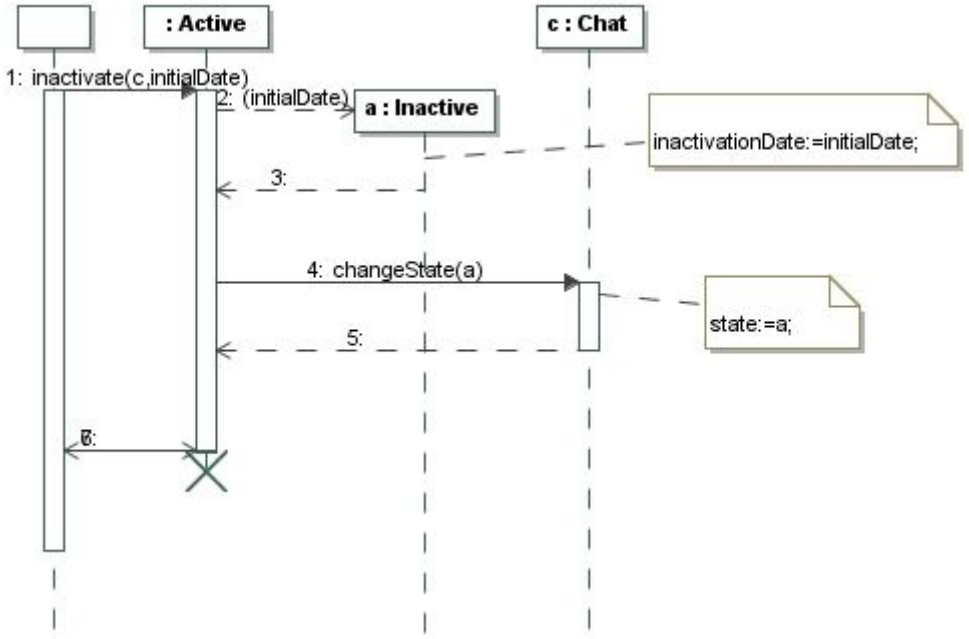
a) Com diu l'enunciat el comportament d'un xat inactiu és diferent del comportament d'un xat actiu. Per tant, el patró que aplicarem és el patró *Estat*.
b) El diagrama de classes mostra l'aplicació del patró *Estat* a la classe *Chat*. S'han definit dos estats, l'estat actiu i l'estat inactiu.



c) El diagrama de seqüència mostra la definició de l'operació *inactivate* de la classe *Chat*. Aquesta operació invoca a una altra del mateix nom a l'objecte *ChatState* que està associat amb el xat. Aquesta última operació és abstracta i es defineix de forma diferent a les seves subclasses (estat *Actiu* i *Inactiu*).


c : Chat
: ChatState
1: inactivate(initialDate)
2: inactivate(c,initialDate)


: Inactive
1: inacitvate(c,initialDate)
2:



Pregunta 4 (25%)

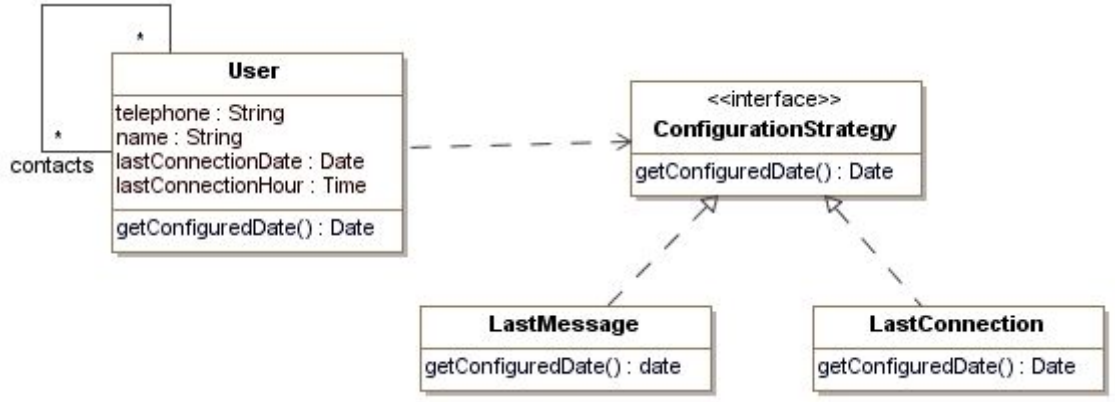
Suposarem ara que volem permetre als usuaris que puguin configurar part de la informació que se'ls hi mostrarà quan entrin al sistema de missatgeria. En concret, la pantalla inicial d'aquest sistema mostrarà a l'usuari tots els xats i, a més, mostrarà o bé la data de l'últim missatge rebut per l'usuari en qualsevol dels seus xats o bé la data de l'última connexió de l'usuari. L'usuari podrà configurar quina data vol que es mostri. Per fer-ho, cada usuari haurà d'indicar la configuració que desitja quan es doni d'alta en el sistema i la podrà canviar sempre que vulgui en temps d'execució. A més, està previst que en el futur es puguin definir altres opcions de configuració que puguin retornar altres dates. Per poder mostrar la informació relacionada amb les configuracions inicials ens demanen dissenyar l'operació *getConfiguredDate(): Date* de la classe *User*. Aquesta operació retornarà la data segons la configuració que tingui associada l'usuari. Es demana:

a) Quin patró de disseny recomanaries per aquest cas (si és que en recomanes algún)?

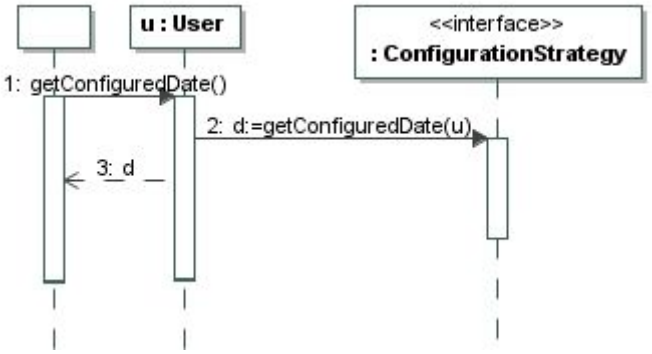
b) Mostra el diagrama de classes resultant d'afegir aquesta funcionalitat.
c) Mostra el diagrama de seqüència o el pseudocodi de l'operació *getConfiguredDate* de la classe *User.* Podeu suposar l'existència de l'operació (que no caldrà proporcionar el diagrama de seqüència) *User::getMessageLastDate():date* que retorna la data del missatge més recent rebut per l'usuari.

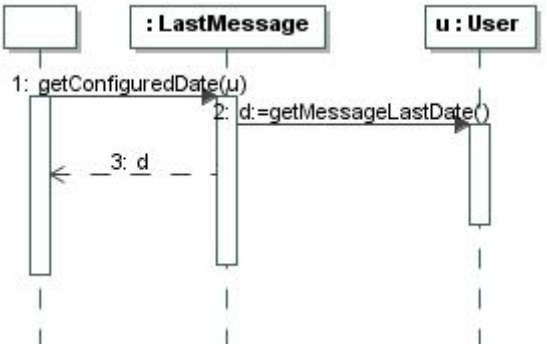
Solució

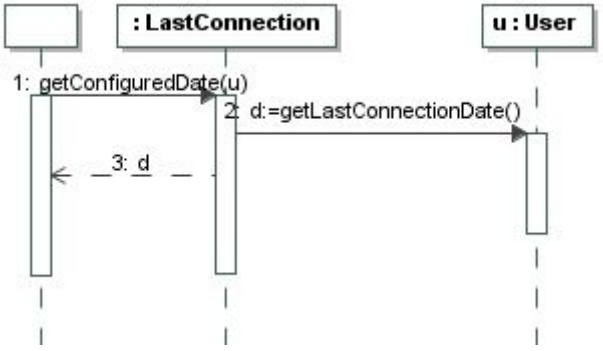
a) Aplicarem el patró *Estratègia,* ja que ens permet definir una família d'algorismes (en el nostre cas les operacions que retornen les dates segons la configuració seleccionada per l'usuari) i permet canviar la configuració en temps d'execució.
b) El diagrama de classes mostra l'aplicació del patró estratègia per representar les dues estratègies (*LastMessage* i *LastConnection*) que permeten mostrar la data segons l'estratègia associada a l'usuari.



c) El diagrama de seqüència mostra la definició de l'operació *getConfiguredDate* de la classe *User*. Aquesta operació invoca a una altra del mateix nom de la interfície*ConfigurationStrategy* que està associada amb l'usuari. Aquesta última operació es defineix de forma diferent a les classes que implementen la interfície.


u : User
<<interface>>
: ConfigurationStrategy
1: getConfiguredDate()
2: d:=getConfiguredDate(u)
3: d


: LastMessage
u : User
1: getConfiguredDate(u)
2: d:=getMessageLastDate()
3: d


: LastConnection
u : User
1: getConfiguredDate(u)
2: d:=getLastConnectionDate()
3: d