L'architecture MVC

(MODÈLE-VUE-CONTRÔLEUR OU MODEL-VIEW-CONTROLLER)

Marc Thivolle - Marseille (17 et 18 mars 2016)

Qui suis-je ? D'où viens-je ?

- 65 ans et un peu plus de trente années de développement de progiciels de gestion dans le cadre de sociétés éditrices de logiciels. Aujourd'hui à la retraite.
- Ma mission au cours de toutes ces années : fournir une interface utilisateur riche, intuitive et conviviale pour la gestion de bases de données de volume important.
- Pratique professionnelle de Delphi (version 3) et VFP (essentiellement version 6).
- Lecture, et vaguement écriture, de Java, Python, Perl et C#.
- J'ai un gros handicap. Pour des raisons contractuelles, je ne peux pas présenter le moindre bout de code issu de mes travaux. Mes propos resteront sans illustration.

Objectifs de la session

- Présenter une architecture (re)mise à la mode par la prolifération des environnements de développement Web
- Avec pour objectifs
  - de devenir plus savant
  - d'utiliser quelques patrons de conception
  - de structurer son code pour éviter l'effet spaghetti

Plan de la session

- Sources
- Historique
- Présentation du modèle
- Les quatre étapes de la construction du diagramme MVC
- Un exemple VFP (de la vitesse à l'indépendance)

Sources

- Article « Modèle-vue-contrôleur » de Wikipédia
- L'ouvrage du GOF : « Design Patterns – Catalogue de modèles de conception réutilisables » - Editions Vuibert (2007)
- L'ouvrage de Steven John Metsker et William C. Wake : « Les Design Patterns en Java – Les 23 modèles de conception fondamentaux » - Campus Press (2006)
- L'ouvrage de Laurent Debrauwer : « Design Pattern en Java – Les 23 modèles de conception » - ENI (2013)
- Différents supports de cours ou didacticiels trouvés sur Internet

Historique

- Travaux du norvégien Trygve Reenskaug lors de son passage au PARC (Palo Alto Research Center) de Xerox (https://en.wikipedia.org/wiki/Trygve_Reenskaug)
- Article « The original MVC reports » (10-12-1979) téléchargeable à partir de l'article français de Wikipédia
- Première implémentation : SmallTalk-80 (début des années 1980)
- Un modèle destiné à la conception des interfaces graphique-souris et des frameworks
- Architecture largement antérieure aux patrons de conception (1995 dans l'ouvrage du GOF)

Présentation : le tronc commun

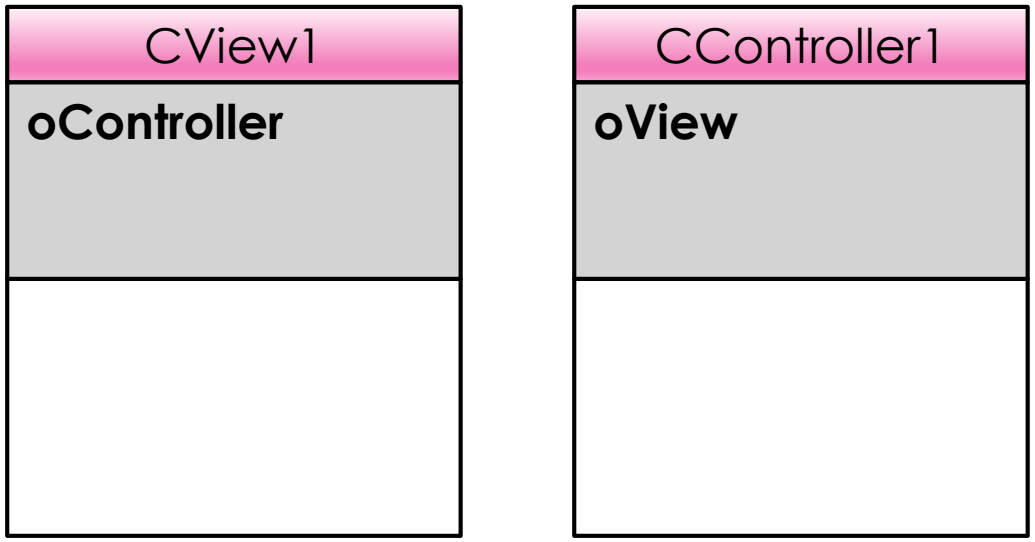
- Accord de tout le monde sur l'existence de trois éléments : le **modèle**, la **vue** et le **contrôleur**
- Accord partiel sur les définitions :
  - le **modèle** : cœur de l'application (base de données, état des données, procédures de consultation et de mise à jour)
  - les **vues** : présentation les données, *interface utilisateur*
  - les **contrôleurs** : logique de contrôle, gestion des évènements, *synchronisation entre les vues et leur modèle*

Présentation : les deux branches

- Là, gros malaise. Une mutation génétique ? Une évolution en deux (ou trois) branches ?
- Article de Chris Adamson.https://community.oracle.com/docs/DOC-983320
- Les deux branches :
  - forme lourde : les **vues** collectent les requêtes utilisateurs et les adressent aux **contrôleurs** qui en assurent la gestion
  - forme légère : les **contrôleurs** implémentent le traitement complet des requêtes utilisateurs
  - forme légère radicale : les **vues** ne communiquent jamais avec le **modèle** (c'est le **contrôleur** qui fabrique les **vues)**

Première étape : une vue, un *ou plusieurs* contrôleurs

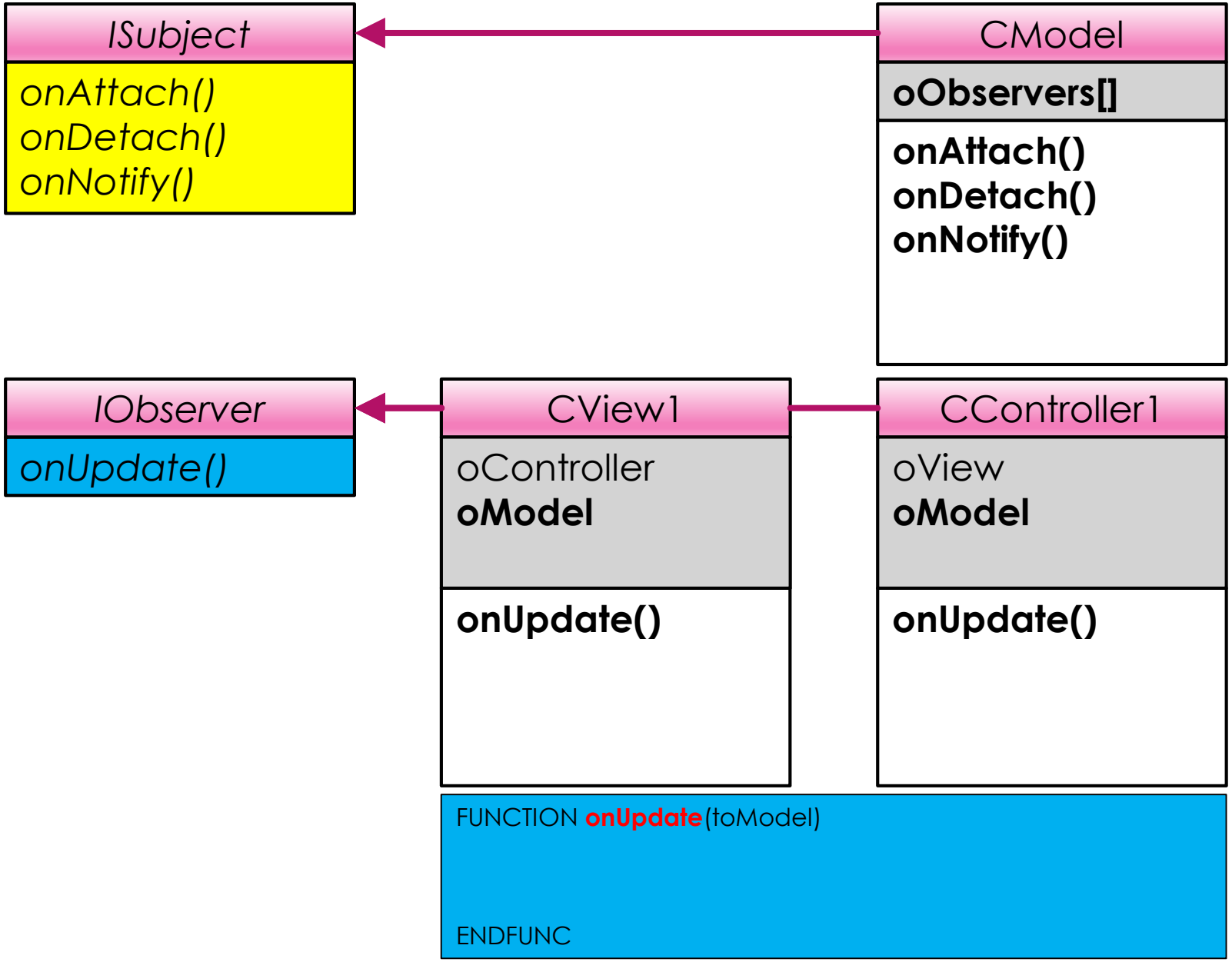
- Un **contrôleur** et un seul va gérer les rapports entre une **vue** et un **modèle**
- La **vue** doit connaitre son **contrôleur** : propriété **oController** (relation de **composition**)
- Par commodité, propriété **oView** dans le **contrôleur**
- La **vue** peut utiliser des **contrôleurs** différents (voir le patron **stratégie**)
  - Différents modes de calcul en fonction des options choisies par l'utilisateur



Une vue et son contrôleur

Deuxième étape : un modèle, des vues, des contrôleurs

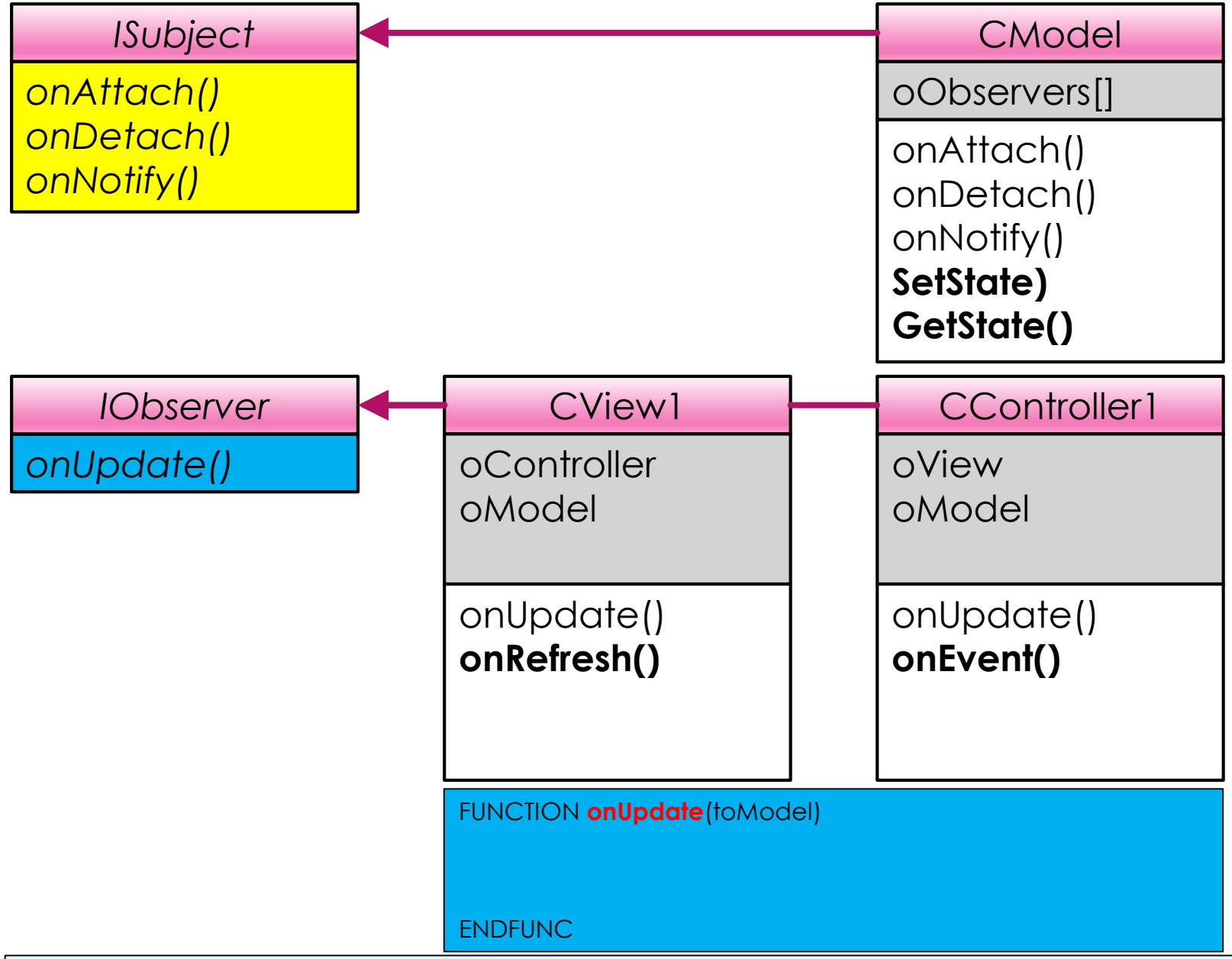
- Plusieurs **vues** peuvent exposer les données (ou l'état) d'un **modèle**
- **Vues** et **contrôleurs** doivent observer le modèle
- Le **modèle** notifie ses modifications aux **vues** et **contrôleurs**
- Patron **observateur**
- Par commodité, propriété **oModel** dans les **vues** et les **contrôleurs**



Le patron observateur

Troisième étape : il faut bien faire quelque chose

- Introduction des méthodes « métier »




```
FUNCTION onAttach(oObserver)
  THIS.oObservers.ADD(oObserver,oObserver.NAME)
ENDFUNC
```

```
FUNCTION onDetach(oObserver)
  THIS.oObservers.REMOVE(oObserver.NAME)
ENDFUNC
```

```
FUNCTION onNotify()
  FOR EACH loObj AS IObserver IN THIS.oObservers
    loObj.onUpdate(THIS)
  ENDFOR
ENDFUNC
```

```
FUNCTION SetState()
  THIS.onNotify()
ENDFUNC
```

```
FUNCTION GetState()
ENDFUNC
```

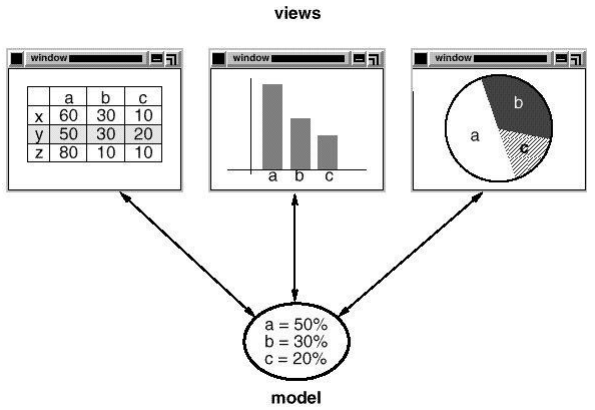
```
FUNCTION onEvent()
  THIS.oModel.SetState()
  THIS.oView.onRefresh()
ENDFUNC
```

```
FUNCTION onRefresh()
ENDFUNC
```

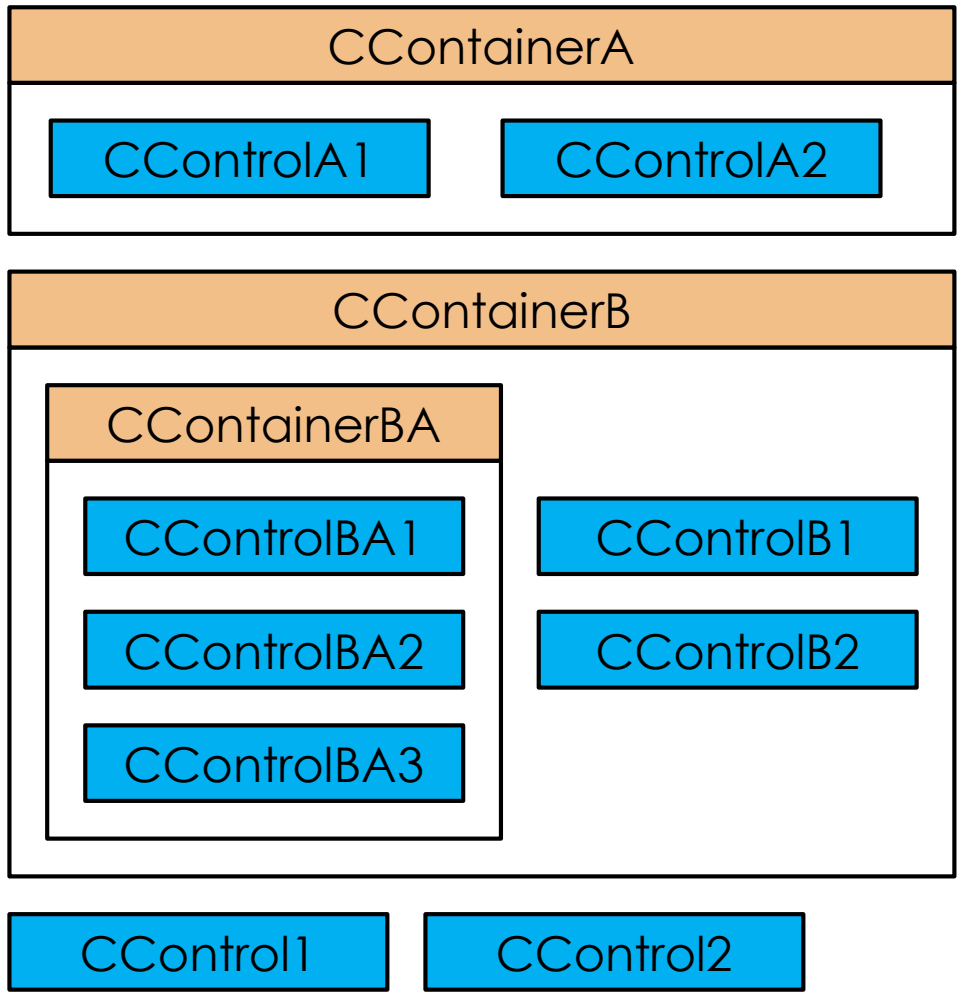
Et il faut bien que tout cela fasse quelque chose.

Quatrième étape : des vues et des contrôleurs complexes

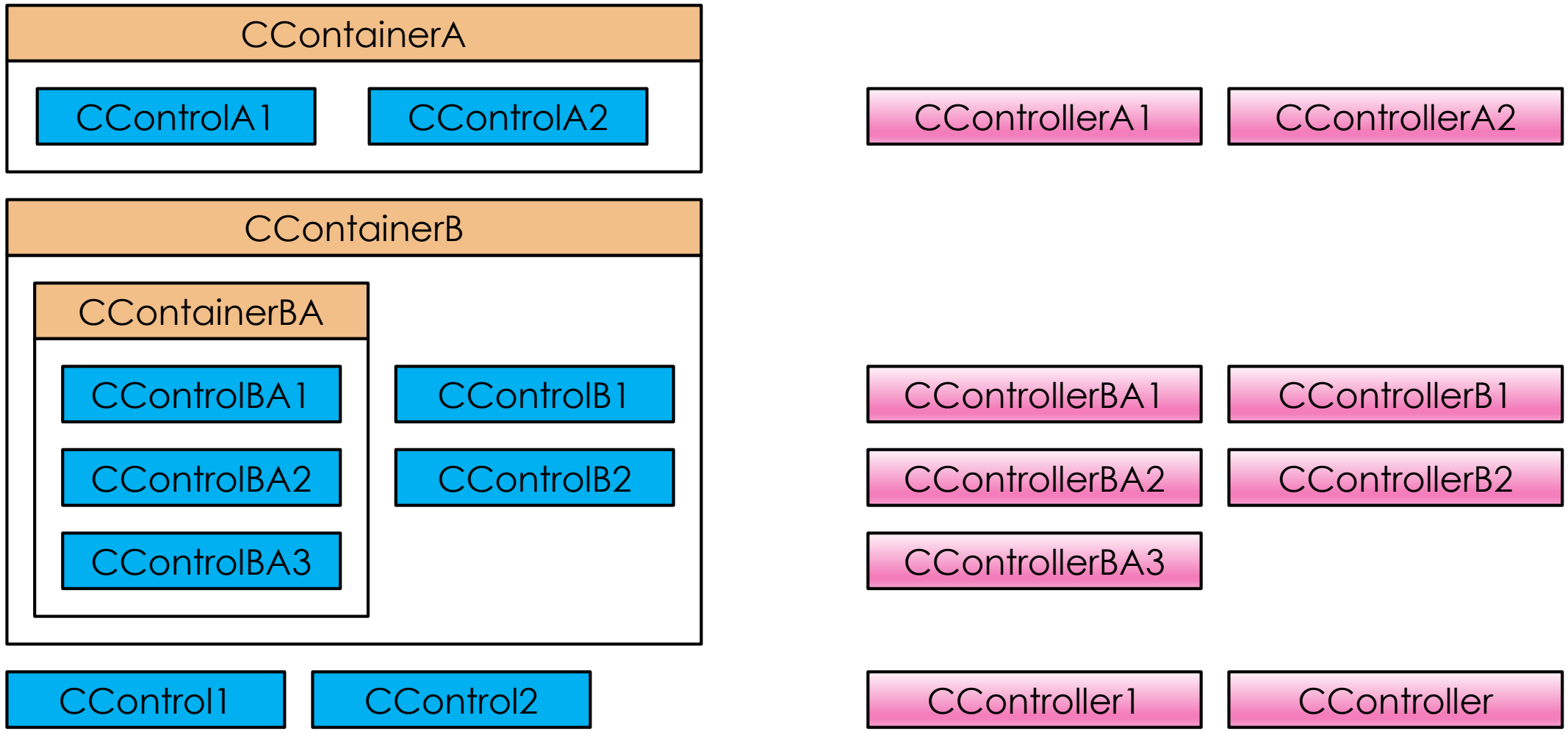
- Une **vue** = un **contrôle** et une (ou quelques) **donnée**(**s**)
- Exemple donné dans l'ouvrage du GOF



- Un formulaire est toujours composée de plusieurs contrôles visuels organisés en containers



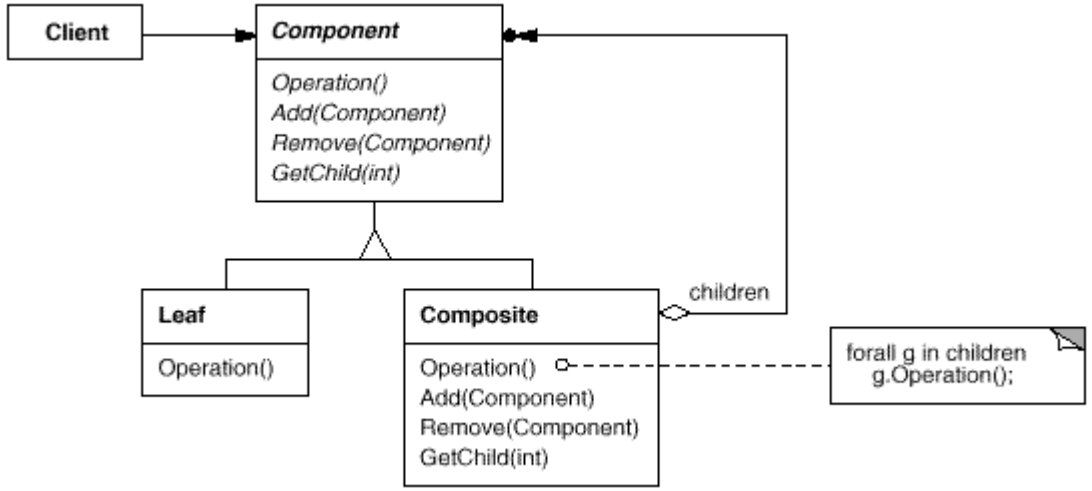
Un formulaire : plusieurs vues

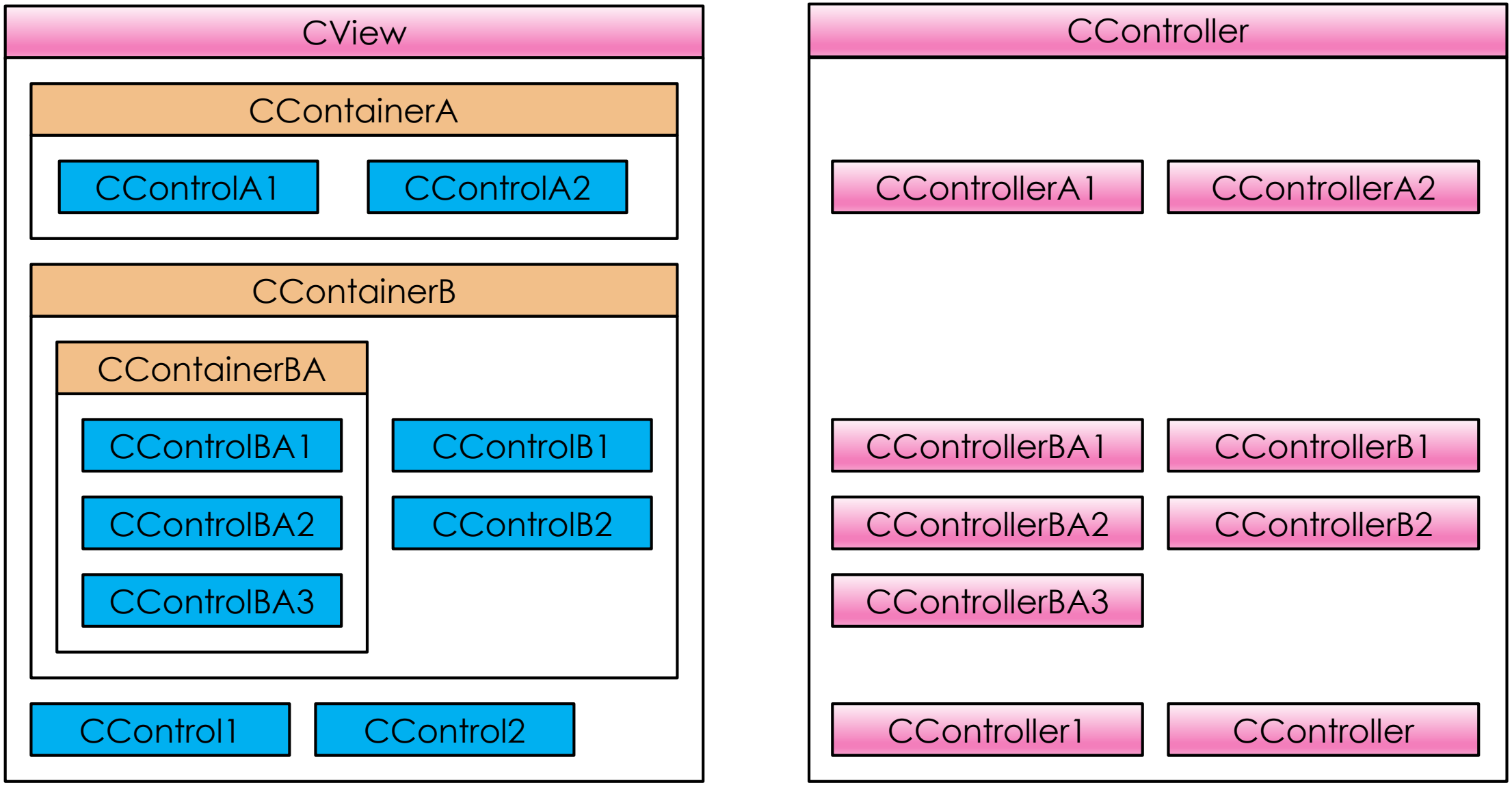


Plusieurs vues, plusieurs contrôleurs

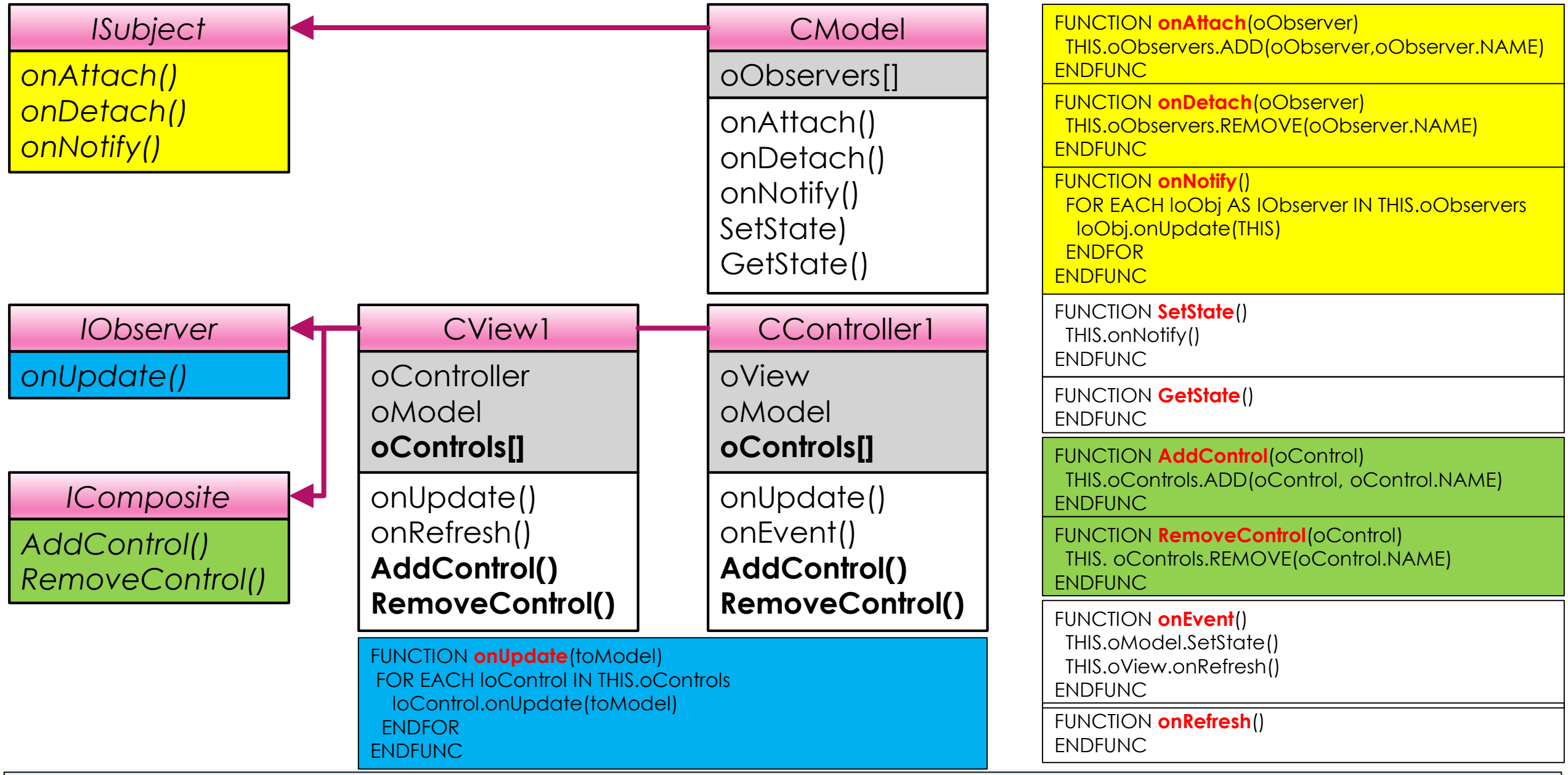
Quatrième étape : des vues et des contrôleurs composites

- Le **modèle** doit notifier un seul objet **vue**
- Le **modèle** doit notifier un seul objet **contrôleur**
- La relation **vue-contrôleur** doit rester multiple : un **contrôleur** par *container* ou contrôle. Un ***super contrôleur*** englobant pour la liaison avec le **modèle**.
- La **vue** et le **contrôleur** sont des objets de type « **composite** »





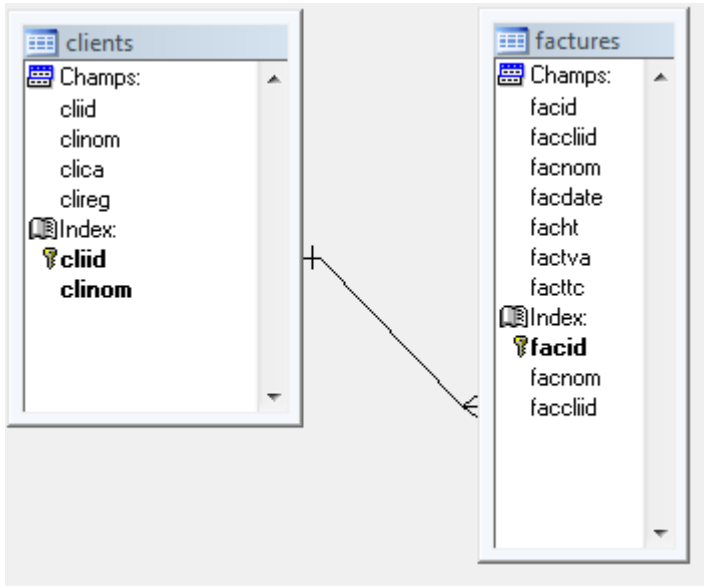
Vue et contrôleur sont des objets composites



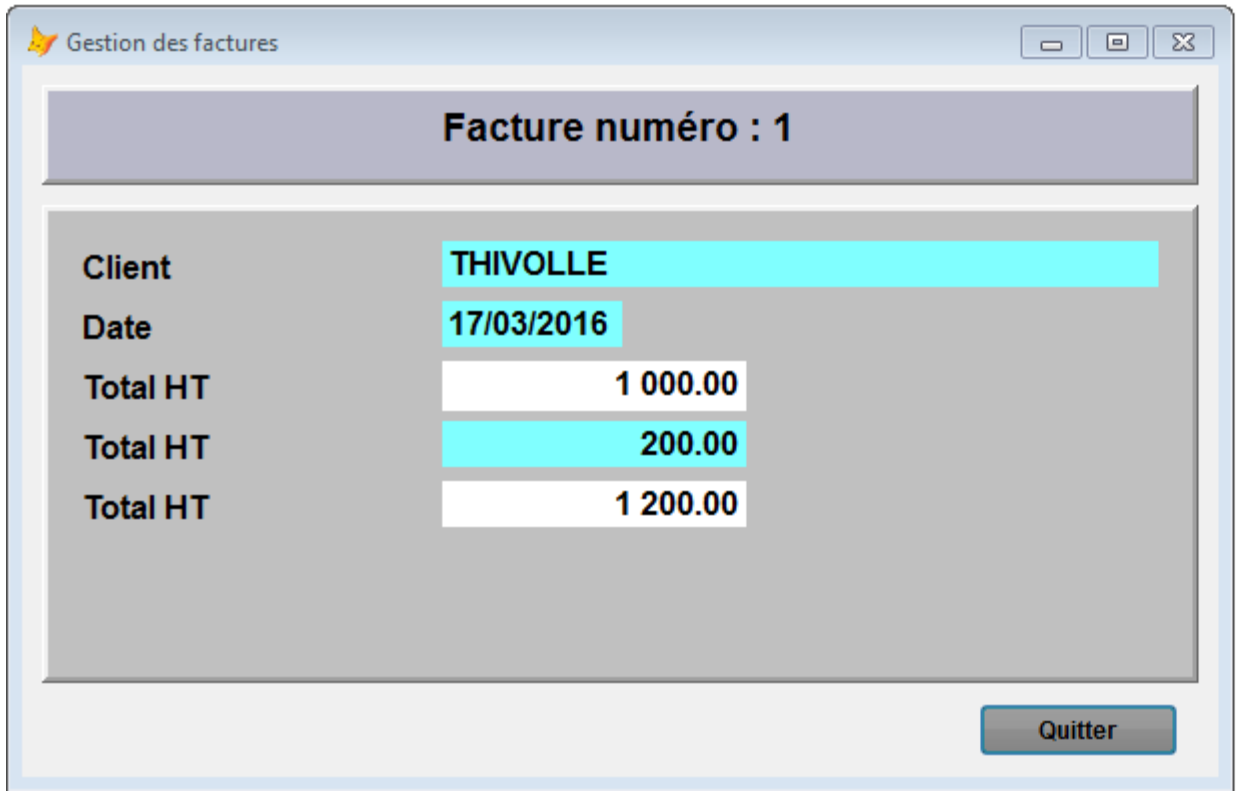
Le diagramme final !

Et VFP dans tout ça ?

- Considéré comme un framework, VFP implémente certains des concepts de l'architecture MVC (branche client lourd)
  - Des contrôles visuels qui permettent d'exposer des données
  - Des contrôles non visuels et/ou tournés vers les données qui permettent de construire un modèle de données
  - Un formulaire qui sert de *container* à la fois pour les contrôles de type vue et leurs contrôleurs (le code écrit dans le SCX)
  - Un lien très fort entre les contrôles et les données, mais malheureusement dans un seul sens : le Refresh. Manque la notification.
- Mais, comme souvent, VFP mélange *in fine* tous les niveaux et autorise les dépendances les plus dangereuses.

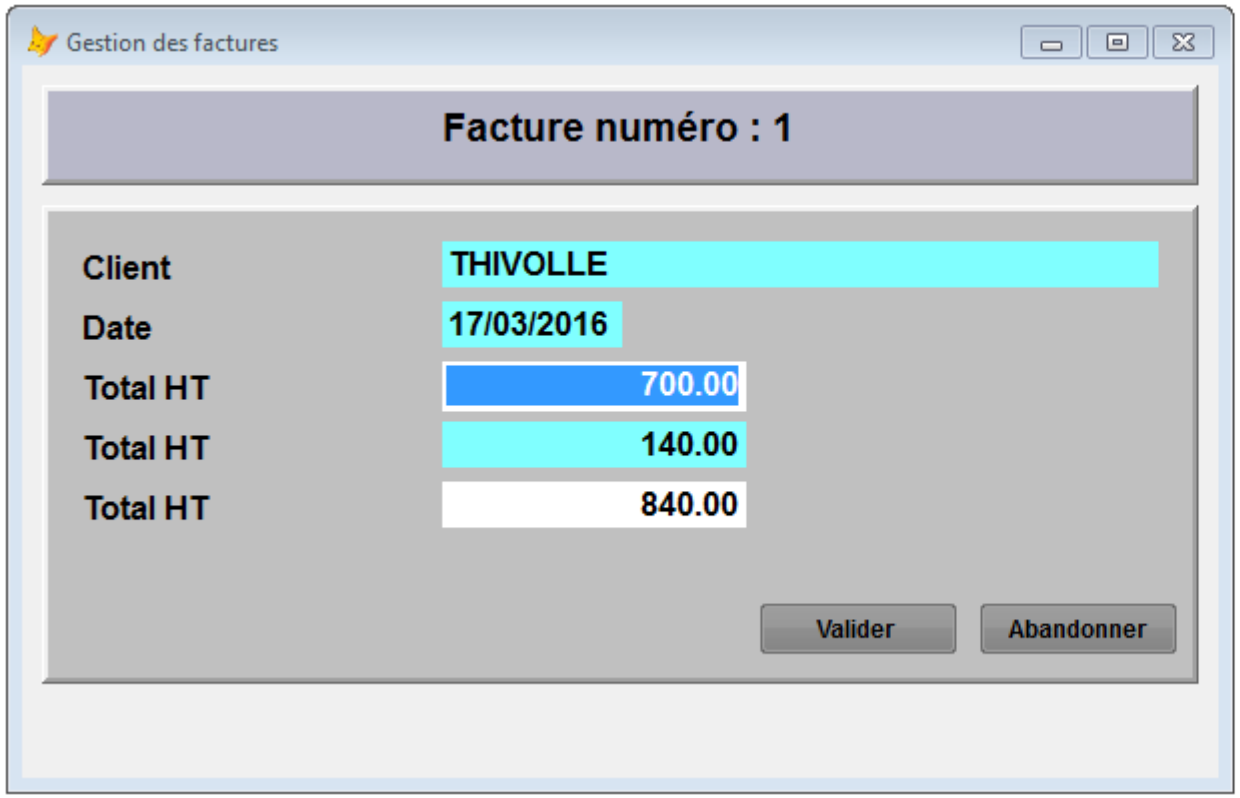


Une application simple : des clients et des factures

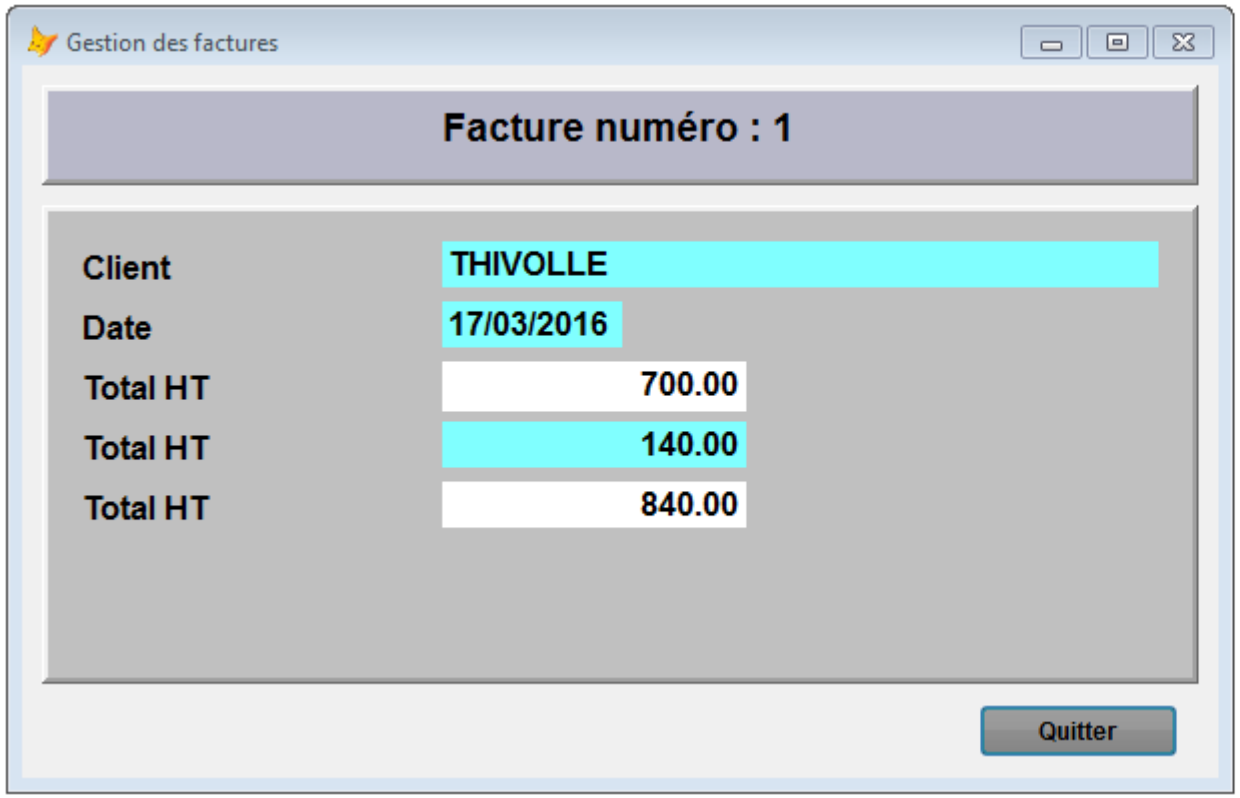

Gestion des factures
Facture numéro : 1
Client
THIVOLLE
Date
17/03/2016
Total HT
1 000.00
Total HT
200.00
Total HT
1 200.00
Quitter


Gestion des clients
Client : THIVOLLE
Chiffre d'affaire
1 200.00
Total des règlements
500.00
Solde
- 700.00
Quitter

Deux écrans



Modification de la facture



Mise à jour du client

De nombreuses dépendances

THISFORM.**container1.txtSolde**.VALUE=**clients.clireg**-**clients.clica**

THIS.**PARENT.cmdValid**.VISIBLE= .T.

REPLACE factures.facttc WITH ROUND(THIS.VALUE*1.20,2)

THIS.**PARENT.txtfacttc**.REFRESH()

lcSql='Update clients set clica=clica+18 where clinom="'+ALLTRIM(factures.facnom)+'"'

THISFORM.**noldfacttc**=**factures.facttc**

goCli.REFRESH()

THISFORM.lModif=.T.

Une gestion unique des procédures d'affichage

Dans tous les objets : **onUpdate**, **onRefresh** (pourquoi deux procédures ?)

Dans les containers

```
FOR EACH loControl IN THIS.CONTROLS
        IF PEMSTATUS(loControl,'onUpdate',5)
        loControl.onUpdate()
    ENDIF
ENDFOR
```

Dans la plupart des objets

onUpdate : this.refresh ou tout autre instruction touchant le contenu

onRefresh : this.visible = .T. ou tout autre instruction touchant le visuel

Les procédures **Refresh** sont remplacées par des appels à **onUpdate** ou **onRefresh** suivant le contexte

Introduction des objets composite et des observateurs

Les dépendances supprimées

~~THISFORM.**container1.txtSolde**.VALUE=**clients.clireg-clients.clica**~~

~~THIS.**PARENT.cmdValid**.VISIBLE= .T.~~

REPLACE factures.facttc WITH ROUND(THIS.VALUE*1.20,2)

~~THIS.**PARENT.txtfacttc**.REFRESH()~~

lcSql='Update clients set clica=clica+18 where clinom='''+ALLTRIM(factures.facnom)+''''

THISFORM.**noldfacttc**=**factures.facttc**

goCli.REFRESH()

THISFORM.lModif=.T.

Une gestion déportée des opérations tables

Un objet **tblCli** : procédure **onMajCa**

Un objet **tblFac** : procédures **onCalcul, onValide, onAbort**

Ces deux objets sont des sujets dans un patron **Observateur** : procédures **Init**, **onAttach**, **onDetach**, **onNotify**

Par commodité, propriété **oModel** dans les **formulaires (**conteneur de **vues)**

Appels **onUpdate** remplacés par un appel à **onNotify** dans les procédures de mise à jour des données

Les dépendances supprimées

~~THISFORM.**container1.txtSolde**.VALUE=**clients.clireg clients.clica**~~

~~THIS.**PARENT.cmdValid**.VISIBLE= .T.~~

~~**REPLACE factures.facttc** WITH ROUND(THIS.VALUE*1.20,2)~~

~~THIS.**PARENT.txtfacttc**.REFRESH()~~

~~**lcSql='Update clients set clica=clica+18 where clinom='''+ALLTRIM(factures.facnom)+''''**~~

~~THISFORM.**noldfacttc=factures.facttc**~~

goCli.REFRESH()

THISFORM.lModif=.T.

Une gestion déportée des opérations tables (premier scénario)

Les deux objets sont dans l'écran des factures.

Dépendance de l'écran des clients : son attachement se fait dans celui des factures. L'écran client observe un sujet présent dans l'écran des factures !

Dépendance des deux objets « données » : **tblfac** utilise **tblcli** en passant par une référence dans le formulaire !

Introduction des objets « données »

Une gestion déportée des opérations tables (deuxième scénario)

Un objet composite dans l'écran des factures (dépendance pour l'écran des clients)

La dépendance de l'écran des clients est conservée.

La dépendance entre **tblcli** et **tblfac** est supprimée en surchargeant la méthode fautive dans le container.

Une gestion déportée des opérations tables (troisième scénario)

Un objet composite au même niveau que les formulaires (le programme appelant) dans une relation de **composition** avec les deux écrans.

La dernière dépendance est supprimée.

Nous avons ici le **modèle** (à la sauce VFP) tant attendu.

Reste à gérer le cas de la propriété lModif du formulaire. Elle doit devenir l'un des états du modèle.

Introduction du modèle (2)

Et les contrôleurs dans tout ça ?

Pourquoi MVC ?

- Parce que, à une lettre près, ça ressemble à ça.