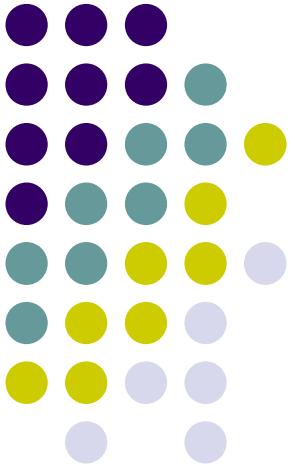
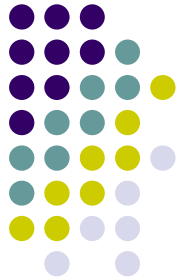
Programmation sans code

Ph.Larvet

Juin 2017 – Juillet 2019

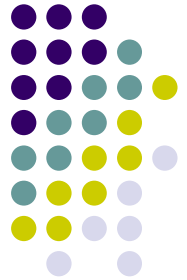


L’approche « Programmation sans code »



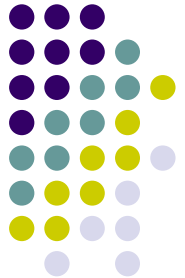
- Dans cette approche, on n’écrit plus de code, mais des COD (*conditions on data* = conditions sur les données).
- Il s’agit d’un mode de programmation purement « déclaratif ».
- On ignore volontairement l'aspect procédural pour se concentrer uniquement sur les données.

Une approche centrée sur les données



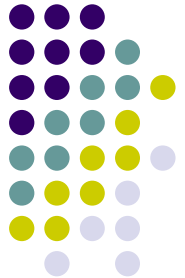
- Quelles sont les DONNEES du problème ?
- A quels OBJETS ces données se rattachent-elles ?
- Quelles sont les CONDITIONS sur ces données ?

Description des données (1)



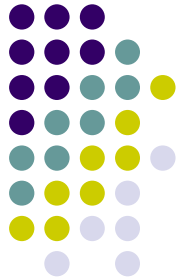
- Les données sont décrites sous forme de couples variable-valeur.
- Les valeurs peuvent être numériques ou booléennes, ou encore représenter la valeur d'une autre variable.
- Par exemple :
  - `temperature = 18`
  - `montant_retrait = 200`
  - `vanne_ouverte = vrai`
  - `resultat = nb1`

Description des données (2)



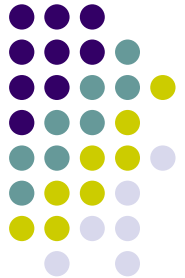
- On peut aussi exprimer une valeur à partir d'un calcul portant sur d'autres variables.
- Par exemple :
  - `valeur_billet = 50`
  - `nombre_billets_distribues = montant_retrait / valeur_billet`
  - `E = m * c²`

Description des COD



- Les conditions sur les données (« COD ») peuvent être vues comme des « Règles » sur les données.
- Elles s'écrivent sous la forme d'implications mathématiques.
- Par exemple :
  - `resultat < limite => nb1 = nb1 + 1`
  - `vanne_ouverte = vrai => reaction_possible = vrai`

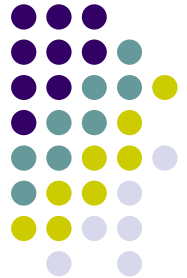
Structure des COD



- La partie gauche de l'implication est appelée « prémisse », la partie droite « conclusion ».
- Une COD peut comporter plusieurs prémisses et / ou plusieurs conclusions.
- Elles sont alors simplement reliées par la conjonction « ET » (ou « AND »).
- Par exemple :

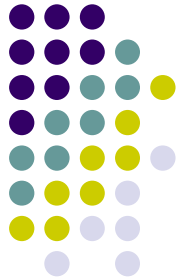
```
carte_introduite = vrai AND code_usager = code_carte =>
retrait_possible = vrai
```

La démarche « Programmation sans code »



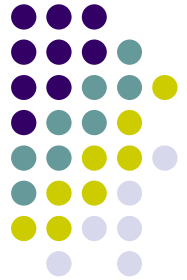
- Le principe et la notation utilisée sont donc très simples.
- La difficulté principale de l’approche est d’exprimer le problème sous forme de données et de COD, rattachées à des OBJETS.

Détail de la démarche



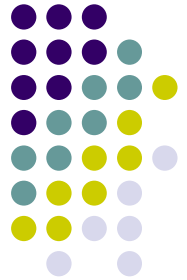
- La démarche comprend trois étapes :
  - - identifier les objets du problème, ou les abstractions qu'on manipule
  - - définir les données liées à ces objets
  - - définir les conditions logiques ou calculatoires liées à ces données.
- Données et COD sont exprimées sous forme de clauses déclaratives, qu'on peut appeler « langage COD ».

Intérêt de l'approche « Programmation sans code »



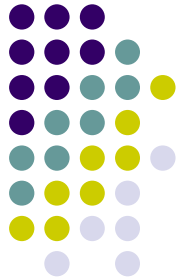
- Le grand intérêt de l'approche est que, les COD étant des relations logiques entre les données des objets, elles peuvent être écrites dans un ordre quelconque.
- En effet, les conditions sur les données ne seront vraies (= vérifiées) que lorsque les données correspondantes auront atteint les valeurs nécessaires.

Environnement de développement et simulation



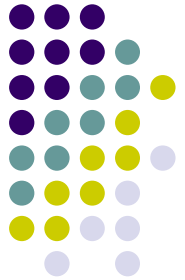
- Un outil logiciel, fonctionnant comme un simulateur, a été développé afin d'exécuter le « langage COD ».
- Cet outil comporte principalement un moteur d'inférence.
- Le moteur examine systématiquement toutes les COD et détermine si les prémisses sont vérifiées : est-ce que les valeurs des variables correspondent aux valeurs présentes dans les différents objets ?

Déclenchement des conclusions



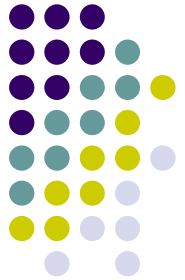
- Si les prémisses sont vérifiées, le moteur tire les conclusions : il affecte de nouvelles valeurs aux données, selon la clause présente dans la conclusion de la COD.
- Le moteur tourne ainsi jusqu'à tirer toutes les conclusions possibles.
- Lorsque plus aucune nouvelle conclusion ne peut être tirée le moteur s'arrête : la simulation est terminée et le résultat final est atteint.

Développement d’un exemple



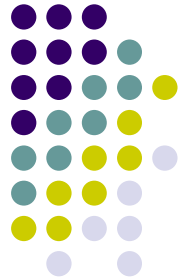
- Nous proposons de développer un cas d’école très simple : le retrait d’espèces sur un distributeur à billets, à l’aide d’une carte bancaire.
- L’usager doit indiquer le « bon » code de sa carte et préciser le montant de son retrait.
- Le retrait n’est pas possible si le code entré est erroné ou si le montant demandé dépasse le montant enregistré sur la carte ou le solde du compte bancaire de l’usager.

Première étape : OBJETS



- Dans le problème (volontairement simplifié) du DAB, les objets sont les suivants :
  - l'usager
  - la carte qui va permettre le retrait
  - le Distributeur lui-même (DAB)
  - le compte bancaire de l'usager (notamment pour autoriser le retrait si le montant demandé ne dépasse pas le solde du compte)

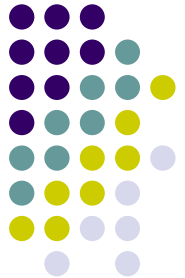
Deuxième étape : DONNEES



- La carte contient son code secret (ici, par exemple 1234) ainsi que le montant maximum de retrait autorisé par semaine (par exemple 500)
- On écrira simplement, en langage COD :

```
CARTE
   code_carte = 1234
   montant_autorise = 500
//----------------------- (ligne commentaire)
```

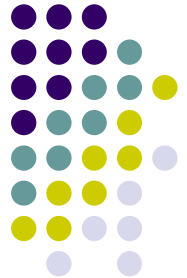
DONNEES (2)



- L’usager va introduire sa carte, entrer son code et le montant souhaité de son retrait.
- On écrira en langage COD :

```
USAGER
   carte_introduite = vrai
   code_usager = 1234
   montant_retrait = 200
//-----------------------------------
```

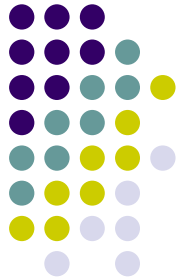
DONNEES (3)



- Le compte bancaire de l'usager possède le montant du solde.
- On écrira :

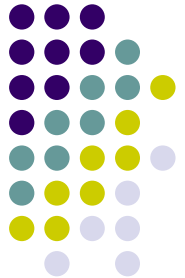
```
COMPTE
   solde_compte = 150
//---------------------------------
```

DONNEES (4)



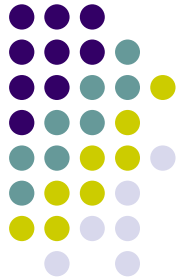
- Le DAB est l'objet principal, responsable du « contrôle » du fonctionnement.
- Il contiendra donc des données qui seront mises à jour au fur et à mesure de l'avancement de la simulation.
- Au départ, ces données sont non valorisées : elles ont la valeur vide « idle ».

DONNEES (5)



```
DAB
   montant_retrait_autorise = idle
   retrait_possible = idle
   retrait_autorise = idle
//-----------------------------------
```

Troisième étape : les COD



- C’est l’étape la plus délicate, qui amène souvent à revoir les deux précédentes.
- L’approche « programmation sans code » est itérative, par essais-erreurs, ce qui finalement est assez naturel…

Conditions sur les données (2)

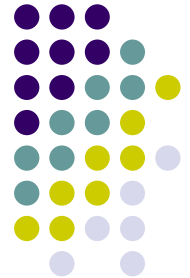
- Ici, le problème est volontairement simplifié.
- Dans la réalité, la carte est lue par un lecteur de carte et c’est à ce niveau que se fait le contrôle de la puce de la carte et du code secret, l’automate du DAB ne se mettant en marche que si le code est OK.

Conditions sur les données (3)

- Dans notre cas simplifié, on considère que, une fois la carte introduite, si le code entré est erroné, le retrait n'est pas possible.
- Ce qui se traduit par la règle suivante, attachée au DAB (le symbole « != » signifie « différent de ») :

```
carte_introduite = vrai AND code_usager != code_carte =>
retrait_possible = faux
```

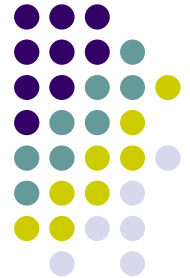
Conditions sur les données (4)



- Il est nécessaire d’écrire la règle inverse, pour que le système soit complet et puisse tourner :

```
carte_introduite = vrai AND code_usager = code_carte =>
retrait_possible = vrai
```

Conditions sur les données (5)



- La règle précédente valorise la donnée interne du DAB « retrait_possible », qui concerne la possibilité du retrait au niveau du code de la carte.
- Il faut à présent vérifier que le montant demandé par l'utilisateur est autorisé, par rapport au montant maxi autorisé par la carte elle-même.
- Ceci s'écrit avec la règle suivante :

```
retrait_possible = vrai AND montant_retrait < montant_autorise =>
montant_retrait_autorise = vrai
```

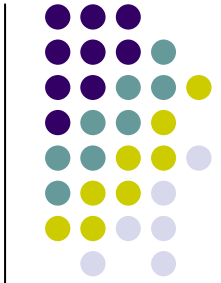
Conditions sur les données (6)

- Il faut enfin vérifier que le montant demandé est autorisé, par rapport au solde du compte.
- Ceci s'écrit par les règles suivantes :

```
montant_retrait_autorise = vrai AND montant_retrait <
solde_compte => retrait_autorise = vrai
montant_retrait_autorise = vrai AND montant_retrait >
solde_compte => retrait_autorise = faux
```

- C'est la donnée interne du DAB « retrait_autorisé » qui déclenchera (ou non !) au final la distribution physique des billets.

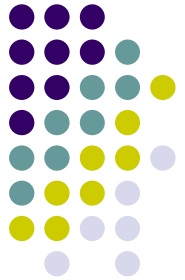
Description complète de l'objet DAB



```
DAB
   montant_retrait_autorise = idle
   retrait_possible = idle
   retrait_autorise = idle
//
   carte_introduite = vrai AND code_usager < code_carte =>
   retrait_possible = faux AND FIN
   carte_introduite = vrai AND code_usager = code_carte =>
   retrait_possible = vrai
//
   retrait_possible = vrai AND montant_retrait < montant_autorise
   => montant_retrait_autorise = vrai
   montant_retrait_autorise = vrai AND montant_retrait <
   solde_compte => retrait_autorise = vrai
   montant_retrait_autorise = vrai AND montant_retrait >
   solde_compte => retrait_autorise = faux AND FIN
//---------------------------------
```

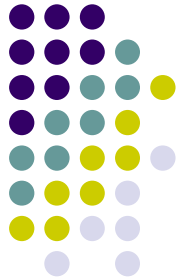
- La clause « FIN » permet de forcer l'arrêt de la simulation.

Compléments de l’exemple



- On peut compléter l’exercice en ajoutant un objet BILLET, auquel on associe la donnée « valeur = 50 » par exemple (si le DAB distribue des billets de 50 €)
- On peut ajouter aussi un autre objet NB_BILLETS, auquel on associe le nombre de billets distribués.

Compléments de l’exemple (2)



- On peut alors ajouter cette règle dans le DAB, qui permet de concrétiser la distribution physique des billets :

```
retrait_autorise = vrai => nb_billets_distribues =
montant_retrait / valeur
```

- En cliquant sur ce lien, vous accéderez à une vidéo présentant une démonstration de cet exemple dans l’environnement « Programmation sans code »

http://www.programmingwithoutcode.org/progsanscode/demo_DAB/