Méthodes et vues

Jan Van den Bussche, Limburg
Emmanuel Waller, Orsay

BDA 98

problème et motivation

1. exécutions *"context-sensitive"* dans un cadre client-serveur (pas forcément BD)[a]

- un client Java exécute un code ; il dispose localement de certaines fonctions mais pas forcément toutes (certaines éventuellement redéfinies par lui)
- au cours de cette exécution, lors d'un appel de fonction, il pourrait souhaiter pour cet appel :
  1. demander au serveur d'exécuter son propre code, *puis* télécharger la valeur du résultat
  2. télécharger le code serveur, *puis* l'exécuter sur place (car il pourra alors utiliser ses propres fonctions client redéfinies)
  3. utiliser son code client (s'il existe)

comment spécifier ces 3 situations :

contexte et mode d'exécution d'un appel ?

[a] d'après un exemple proposé par S. Abiteboul

motivation BD

2. vues à objets

- base racine $\equiv$ serveur

  vue $\equiv$ client

- – lors de l'exécution d'une méthode de la vue, comment spécifier le contexte et le mode d'exécution des appels de méthodes imbriqués ?
  – de plus, dans le cas du "téléchargement" du code de la base racine, comment spécifier statiquement des choix de résolution différents pour des appels imbriqués ?

même question que dans cadre client-serveur

(rq : on ne considère pas les mises à jour de la vue)

contexte : à objets

- BD = schéma + instance
  = hiérarchie de classes + attributs + méthodes + objets + références
- méthodes = héritage + redéfinition + liaison dynamique

ex : O_2, etc.

exemple - BD et vue

- BD d'une compagnie d'assurance contient

```
classe Client
attributs ville, age, risque
methode prime = age + risque
```

- la compagnie envisage pour les parisiens
  - un risque spécifique
  - un contrat promotionnel

  pour étudier cette situation, elle crée une *vue comportementale* qui contient

```
classe Parisien
population ...[a]  /* les parisiens */
attributs age /* tel quel */
          risque = 20
methodes prime = age * risque
         promo_prime = 50 % prime
```

[a] spécifié par un mécanisme structurel quelconque de la littérature, par exemple O_2Views

exemple - exécution : problème

- instance
  $Client\ o : Paris, age\ 42, risque\ 11$
- exécution lancée dans la vue
  $promo\_prime(o)$
  $\rightarrow mult(0.5, prime(o))$
  $\rightarrow mult(0.5, ???)$
- question
  dans quel contexte et comment doit être exécuté $prime(o)$???
  - calcul dans vue ?
  - calcul dans base ?
  - calcul dans vue avec code base ?

3 choix $\Rightarrow$ 3 résultats différents !!!

voyons comment et pourquoi...

exemple - 3 résultats différents

- o : $age\ 42, risque\ 11$; Parisien : risque $= 20$; prime : age * risque ; promo_prime : 50 % prime
  $promo\_prime(o) \rightarrow mult(0.5, prime(o))$

1. calcul de prime dans la vue
   $\overset{vue}{\rightarrow} mult(0.5, mult(age(o), risque(o)))$
   $\overset{vue}{\rightarrow} mult(0.5, mult(42, 20))$
   $\overset{vue}{\rightarrow} mult(0.5, 840) \overset{vue}{\rightarrow} 420$

2. dans base racine
   $\overset{BD}{\rightarrow} mult(0.5, somme(age(o), risque(o)))$
   $\overset{BD}{\rightarrow} mult(0.5, somme(42, 11))$ (risque base rac.)
   $\overset{BD}{\rightarrow} mult(0.5, 53) \overset{vue}{\rightarrow} 26.5$

3. dans vue avec code base racine
   $\overset{BD}{\rightarrow} mult(0.5, somme(age(o), risque(o)))$
   $\overset{vue}{\rightarrow} mult(0.5, somme(42, 20))$ (risque vue)
   $\overset{vue}{\rightarrow} mult(0.5, 62) \overset{vue}{\rightarrow} 31$

le programmeur doit pouvoir choisir

exemple - avantages et problèmes

- avantages (exemples)
  - vue ordinaire : un client emménage à Paris
    $\Rightarrow$ il apparaît automatiquement dans la vue avec la valeur correcte pour risque
  - *vue comportementale* :
    la formule de calcul de la prime change
    $\Rightarrow$ la vue s'adapte automatiquement
- problèmes (exemples)
  - comment *spécifier* contexte et mode d'exécution de chaque appel ?
  - comment vérifier la *correction* de cette formule : jamais d'appel de méthode indéfini (vue ou base racine) ?

vue

BD racine
schéma (fixe)
+

spécification de la vue

vue
schéma (fixe)
+

modifications base racine répercutées dans vue

définition et motivation

vue ordinaire

$=$ classes $+$ attributs

doivent dépendre

de ceux

de la base racine

définition et motivation

vue comportementale

$=$ classes $+$ attributs $+$ méthodes

doivent dépendre
de celles
de la base racine

$\Rightarrow$

- schéma racine $=$
  classes $+$ signatures pptés (stockées/calculées)
- instance racine $=$ objets $+$ références $+$ codes

vues à objets : structurel (assez) bien connu

comportemental : très peu

travaux antérieurs

O_2Views : thèse Souza +
Souza, Abiteboul, Delobel, EDBT 94

(programmation E. Waller : appli pour revue Esprit)

- axé sur l'aspect structurel
- – méthodes : appelées par requêtes lors calcul attributs virtuels
  – sémantique : ça tourne parfaitement, mais... implicite, ad hoc, inconfortable...

A formal model of views : Guerrini et al, TAPOS 97

- axé sur l'aspect structurel
- – méthodes (règles déductives) : possible disposer dans la vue d'un code de la base racine, mais...
  – sémantique : non fournie...

le programmeur ne peut pas choisir

contexte et mode d'exécution d'un appel

nos principaux résultats

- un formalisme[a] pour spécifier le transfert de contrôle entre vue et base lors d'un calcul, avec
  - calcul dans vue
  - calcul dans base
  - calcul dans vue avec code base
- décidabilité (dans certains cas) de la correction

[a] au-dessus du formalisme des schémas de méthodes : thèse Waller + Abiteboul, Kanellakis, Ramaswamy, Waller, JCSS 95

plan

1. introduction
   travaux antérieurs
2. définition du cadre
   (vues comportementales)
3. les primitives pour écrire les méthodes de la vue
   (spécification d'un comportement de vue)
4. vérification de correction
5. conclusion

vue - définition

- input d'une vue

  schéma rac. (fixe) = hiér. class. + sign. pptés

  instance racine = objets + références + *codes*

- output

  schéma ordinaire : comme base racine[a] (fixe)

  instance ordinaire = objets + propriétés

  $\Rightarrow$ ces propriétés : utilisateur final extér. à vue

  - ne sait pas si elles sont stockées ou calculées
  - écrit au-dessus ses propres méthodes codées

- définition

  soient R schéma racine et V schéma (appelé *schéma de la vue*) :

  une *vue* (ou *fonction de vue*) de R dans V est une fonction des instances racines de R dans les instances de V

a view is just a (double) query...

[a] mais bien sûr pas forcément le même

fonction de vue

vue output = population + propriétés :

à spécifier

- population de vue =
  requête (fonction abstraite) pour chaque classe

  (utiliser mécanisme de spécification quelconque de la littérature)

- comportement de vue =
  des codes définissant les propriétés
  déclarées dans le schéma de la vue

plan

1. introduction
   travaux antérieurs
2. définition du cadre
   (vues comportementales)
3. les primitives pour écrire les méthodes de la vue
   (spécification d'un comportement de vue)
4. vérification de correction
5. conclusion

importation de valeur

- exemple de *définition de méthode de vue* :

  ancien_risque@Parisien = *import value risque*

- lors calcul : on récupère la valeur de la propriété *risque* dans la base racine (peu importe qu'elle soit stockée ou calculée)

  exécution du code dans la base racine

notre exemple (variante)

- même base racine

  Client : ville age, risque, prime = age + risque

- vue

  Parisien :

  ancien_risque = import value risque
  risque = 2 * ancien_risque
  nouvelle_prime = import code prime

sémantique de *import code* ?

importation de code

- autre exemple de définition de méthode de vue :
  nouvelle_prime@Parisien = *import code prime*
- lors calcul : on récupère le code de la propriété *prime* dans la base racine, puis on continue l'exécution dans la vue
- $nouvelle\_prime(o)$
  $\overset{vue}{\rightarrow} prime^{\mathbf{ic}}(o)$ (*prime* en import code)
  $\overset{vue}{\rightarrow} somme(age(o), risque^{???}(o))$
  récursivement : quel mode choisir pour *risque* ?
- nouvelle_prime@Parisien =
  import code prime
  with (age : import value, risque : view)
- pour chaque méthode racine, *with* spécifie son mode (voir subtilités dans papier)

+ spécification globale d'importation de code

comportement de vue

- $=$ spécification globale d'import. de code

  $+$

  définitions de méthodes de vue :

  - $m@c = code$ (à exécuter dans la vue)
  - $m@c = import\ value\ m'$
  - $m@c = import\ code\ m'$

- sémantique formelle (voir papier)

résultats d'expressivité

- prop :
  un comportement de vue sans code normaux
  $\equiv$ le même avec uniquement import value
- prop :
  tout comportement de vue
  $\equiv$ un autre sans import value
  (mais crucial conceptuellement et en pratique)

plan

1. introduction
   travaux antérieurs
2. définition du cadre
   (vues comportementales)
3. les primitives pour écrire les méthodes de la vue
   (spécification d'un comportement de vue)
4. vérification de correction
5. conclusion

résultat

- *cohérence* =
  aucune exécution n'aboutit à un appel de méthodes (dans vue ou base racine) indéfini
- thm :
  - la cohérence des vues comportementales se réduit à
    celle des schémas de méthodes ordinaires
  - elle est donc *décidable*
    en particulier pour les cas de
    méthodes unaires et/ou non récursives
- (rq : autres propriétés intéressantes, voir papier)

plan

1. introduction
   travaux antérieurs
2. définition du cadre
   (vues comportementales)
3. les primitives pour écrire les méthodes de la vue
   (spécification d'un comportement de vue)
4. vérification de correction
5. conclusion

conclusion

- résultats
  - un formalisme pour spécifier le contexte et le mode d'exécution des appels de méthode
    - calcul dans vue
    - calcul dans base (*import value*)
    - calcul dans vue avec code base (*import code + spéc. globale*)
  - expressivité des primitives (ébauche)
  - décidabilité (dans certains cas) de la correction
- extensions : variantes de import code et import value, masquage, objets imaginaires (papier)
- questions
  - implémentation et expérimentation
  - "complétude" : import value et import code capturent-elles "tout" ? que veut-on ?...