

Tissage de propriétés

contrôle d'accès, ordonnancement, disponibilité

Pascal Fradet

Inria Rhône-Alpes
Grenoble, France

Réunion ACI DISPO - 22 septembre 2006

Plan

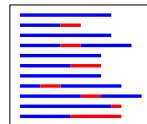
- 1 Éléments de programmation par aspects
- 2 Application au contrôle d'accès
- 3 Application à la fusion de composants
- 4 Application à la disponibilité
- 5 Conclusion

Plan

- 1 Éléments de programmation par aspects
- 2 Application au contrôle d'accès
- 3 Application à la fusion de composants
- 4 Application à la disponibilité
- 5 Conclusion

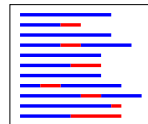
L'idée

- L'implémentation de certains aspects non-fonctionnels traverse le programme
 - par ex. profilage, sécurité, synchronisation
 - abstractions (fonctions, procédures, classes) inadaptées



L'idée

- L'implémentation de certains aspects non-fonctionnels traverse le programme
 - par ex. profilage, sécurité, synchronisation
 - abstractions (fonctions, procédures, classes) inadaptées
- Programmation par Aspects [Kiczales et al. 1997]



Approche syntaxique

- Composant vu comme un *arbre de syntaxe abstraite*

Approche syntaxique

- Composant vu comme un *arbre de syntaxe abstraite*
- Aspect : directives d'insertion de code
 - motifs sur les noms de méthodes, d'opérateurs, etc.
 - motifs sur l'état (valeur des variables)

Approche syntaxique

- Composant vu comme un *arbre de syntaxe abstraite*
- Aspect : directives d'insertion de code
 - motifs sur les noms de méthodes, d'opérateurs, etc.
 - motifs sur l'état (valeur des variables)
- Tissage : une passe pour insérer les actions des aspects

AspectJ

- Langage d'aspect pour Java
 - le précurseur et le plus connu
 - général et expressif

AspectJ

- Langage d'aspect pour Java
 - le précurseur et le plus connu
 - général et expressif
- Exemple

```
aspect ObserverUpdating
{
  pointcut moves() : call(void Point.setX(int))
    || call(void Point.setY(int))
    || call(void Point.setZ(int));

  after() : moves() { Display.update() }
}
```

Approche sémantique

- Composant vu comme ses *traces d'exécution*

Approche sémantique

- Composant vu comme ses *traces d'exécution*
- Aspect : propriété de sûreté
 - ensemble de traces autorisées

Approche sémantique

- Composant vu comme ses *traces d'exécution*
- Aspect : propriété de sûreté
 - ensemble de traces autorisées
- Tissage : moniteur observant l'exécution
 - mise en oeuvre statique plus compliquée

Plan

- 1 Eléments de programmation par aspects
- 2 Application au contrôle d'accès
- 3 Application à la fusion de composants
- 4 Application à la disponibilité
- 5 Conclusion

Application au contrôle d'accès

- Composant : applet

Application au contrôle d'accès

- Composant : applet
- Aspect : politique de contrôle d'accès
 - description des évènements d'intérêts
 - description des traces autorisées
 - propriété régulière (automate à états finis)

Exemple

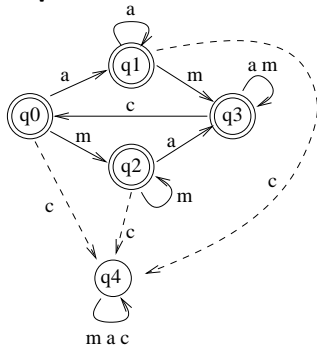
Composant source

```
manager() ;  
if(test1) {accountant() ;}  
if(test2) {critical() ;manager() ;}  
accountant() ;  
critical() ;
```

Évènements

m → manager
a → accountant
c → critical

Aspect

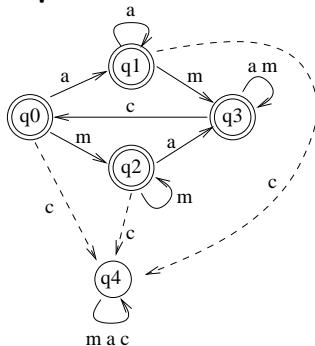


Exemple

Composant tissé

```
s=0 ;  
manager() ;  
if(test1) { s=1; accountant() ;}  
if(test2) { if s=0 then abort  
            critical() ;manager() ;}  
accountant() ;  
critical() ;
```

Aspect



Bilan

- Efficace et modulaire (maintenance)

Bilan

- Efficace et modulaire (maintenance)
- Pas de rejet de programmes

Bilan

- Efficace et modulaire (maintenance)
- Pas de rejet de programmes
- Manque de généralité
 - un seul type d'action (abort)
 - plutôt pour du code mobile

Bilan

- Efficace et modulaire (maintenance)
- Pas de rejet de programmes
- Manque de généralité
 - un seul type d'action (abort)
 - plutôt pour du code mobile
- Objectifs dans l'ACI DISPO
 - généraliser l'approche
 - appliquer aux composants et à la disponibilité

Plan

- 1 Eléments de programmation par aspects
- 2 Application au contrôle d'accès
- 3 Application à la fusion de composants**
- 4 Application à la disponibilité
- 5 Conclusion

Application à la fusion de composants

- Motivations
 - allier modularité et efficacité
 - fusionner les composants
 - éliminer les communications, changements de contexte, etc.

Application à la fusion de composants

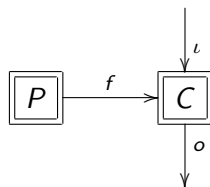
- Motivations
 - allier modularité et efficacité
 - fusionner les composants
 - éliminer les communications, changements de contexte, etc.
- Approche
 - aspects d'ordonnement
 - sélection de traces parmi l'ensemble des entrelacements
 - basée sur les LTS et une opération de produit

Réseaux de composants

- Réseaux de Kahn : expressifs, déterministes, compositionnels

$$P = \left\{ \begin{array}{l} p_0 : x := x + 1 \rightsquigarrow p_1; \\ p_1 : f!x \rightsquigarrow p_0 \end{array} \right\}$$

$$C = \left\{ \begin{array}{l} c_0 : a < b \mid f?a \rightsquigarrow c_1; \\ c_0 : a \geq b \mid \iota?b \rightsquigarrow c_1; \\ c_1 : o!(a + b) \rightsquigarrow c_0 \end{array} \right\}$$

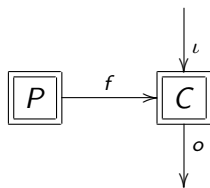


Réseaux de composants

- Réseaux de Kahn : expressifs, déterministes, compositionnels
 - communications point à point asynchrones via FIFO

$$P = \left\{ \begin{array}{l} p_0 : x := x + 1 \rightsquigarrow p_1; \\ p_1 : f!x \rightsquigarrow p_0 \end{array} \right\}$$

$$C = \left\{ \begin{array}{l} c_0 : a < b \mid f?a \rightsquigarrow c_1; \\ c_0 : a \geq b \mid \iota?b \rightsquigarrow c_1; \\ c_1 : o!(a + b) \rightsquigarrow c_0 \end{array} \right\}$$

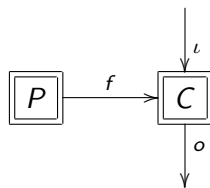


Réseaux de composants

- Réseaux de Kahn : expressifs, déterministes, compositionnels
 - communications point à point asynchrones via FIFO
 - écriture non bloquante et lecture bloquante

$$P = \left\{ \begin{array}{l} p_0 : x := x + 1 \rightsquigarrow p_1; \\ p_1 : f!x \rightsquigarrow p_0 \end{array} \right\}$$

$$C = \left\{ \begin{array}{l} c_0 : a < b \mid f?a \rightsquigarrow c_1; \\ c_0 : a \geq b \mid \iota?b \rightsquigarrow c_1; \\ c_1 : o!(a + b) \rightsquigarrow c_0 \end{array} \right\}$$

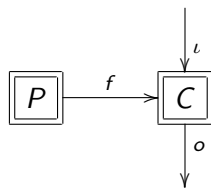


Réseaux de composants

- Réseaux de Kahn : expressifs, déterministes, compositionnels
 - communications point à point asynchrones via FIFO
 - écriture non bloquante et lecture bloquante
 - tests de l'état des files interdits

$$P = \left\{ \begin{array}{l} p_0 : x := x + 1 \rightsquigarrow p_1; \\ p_1 : f!x \rightsquigarrow p_0 \end{array} \right\}$$

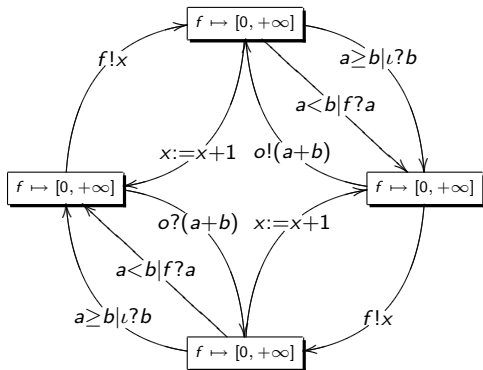
$$C = \left\{ \begin{array}{l} c_0 : a < b \mid f?a \rightsquigarrow c_1; \\ c_0 : a \geq b \mid \iota?b \rightsquigarrow c_1; \\ c_1 : o!(a + b) \rightsquigarrow c_0 \end{array} \right\}$$



Abstraction du réseau de composants

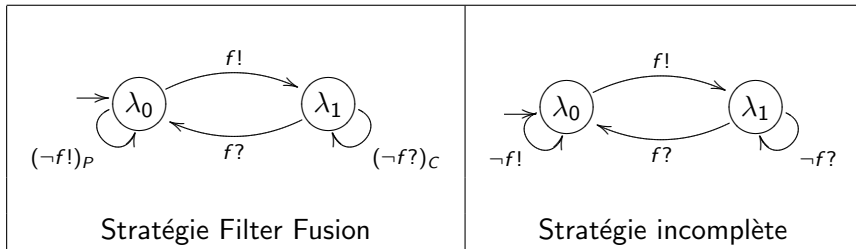
P
 $p_0 : x := x + 1 \rightsquigarrow p_1$
 $p_1 : f!x \rightsquigarrow p_0$

C
 $c_0 : a < b \mid f?a \rightsquigarrow c_1$
 $c_0 : a \geq b \mid \iota?b \rightsquigarrow c_1$
 $c_1 : o!(a+b) \rightsquigarrow c_0$



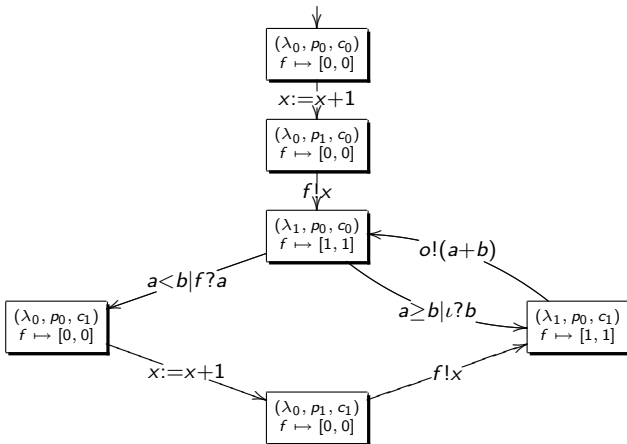
Contraintes d'ordonnancement (aspects)

- LTS gardés à états finis



Tissage

- Produit entre l'abstraction et la contrainte



Bilan

- Autres étapes
 - compléter séquentialisation (en assurant l'équité)
 - concrétisation (produire un unique programme séquentiel)

Bilan

- Autres étapes
 - compléter séquentialisation (en assurant l'équité)
 - concrétisation (produire un unique programme séquentiel)
- Généralisation du cadre précédent
 - au parallélisme (ordonnancement et implémentation efficace)
 - sélection d'une (ou de) traces parmi les possibles

Bilan

- Autres étapes
 - compléter séquentialisation (en assurant l'équité)
 - concrétisation (produire un unique programme séquentiel)
- Généralisation du cadre précédent
 - au parallélisme (ordonnancement et implémentation efficace)
 - sélection d'une (ou de) traces parmi les possibles
- Plus flexible et formel que les approches existantes

Plan

- 1 Eléments de programmation par aspects
- 2 Application au contrôle d'accès
- 3 Application à la fusion de composants
- 4 Application à la disponibilité**
- 5 Conclusion

Application à la disponibilité

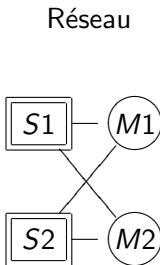
- Motivations
 - assurer l'accès aux ressources
 - prévenir les dénis de service

Application à la disponibilité

- Motivations
 - assurer l'accès aux ressources
 - prévenir les dénis de service
- Approche
 - aspects : politiques de disponibilité (propriétés temporisées)
 - sélection de comportements temporels autorisés
 - basée sur les automates temporisés et l'opération de produit

Cadre

- Services qui accèdent à des ressources partagées



Services

```
S1 =  
l1 : M1.prendre(); M2.prendre();  
S1Calcul;  
M2.relacher(); M1.relacher();  
jmp l1;
```

```
S2 =  
l2 : M2.prendre();  
S2Calcul1;  
if G then M1.prendre();  
S2Calcul2;  
M1.relacher();  
M2.relacher();  
jmp l2;
```

- Possibilités de dénis de service
 - famine si ne termine pas si S2Calcul1 ne termine pas
 - interblocage pour la prise de M1 et M2

Aspects

- Les aspects (les politiques de disponibilité)
 - contraignent la gestion des ressources de chaque service
 - sont exprimés sur l'histoire d'exécution
 - insèrent des minuteurs et des attentes
 - sont tissés sur le code des services

Aspects

- Les aspects (les politiques de disponibilité)
 - contraignent la gestion des ressources de chaque service
 - sont exprimés sur l'histoire d'exécution
 - insèrent des minuteurs et des attentes
 - sont tissés sur le code des services
- Assurer que la ressource M1 est relâchée avant 25 secondes

$a_1 = \text{M1.prendre} \triangleright \text{reset}(i_1, 25); a_2$

$a_2 = \text{M1.relacher} \triangleright \text{cancel}(i_1); a_1$

Aspects

- Les aspects (les politiques de disponibilité)
 - contraignent la gestion des ressources de chaque service
 - sont exprimés sur l'histoire d'exécution
 - insèrent des minuteurs et des attentes
 - sont tissés sur le code des services
- Assurer que la ressource M1 est relâchée avant 25 secondes

$$\begin{aligned} a_1 &= \text{M1.prendre} \triangleright \text{reset}(i_1, 25); a_2 \\ a_2 &= \text{M1.relacher} \triangleright \text{cancel}(i_1); a_1 \end{aligned}$$

- Attendre 20 sec. entre chaque prise de la ressource M

$$\begin{aligned} a_1 &= \text{M.prendre} \triangleright \text{start}(t); a_2 \\ a_2 &= \text{M.prendre} \triangleright \{\text{wait}(t, 20); \text{start}(t)\}; a_2 \end{aligned}$$

Tissage

- Services abstraits en automates temporisés
 - sur approximation des traces d'exécution
 - sur approximation du comportement temporel

Tissage

- Services abstraits en automates temporisés
 - sur approximation des traces d'exécution
 - sur approximation du comportement temporel
- Aspects traduits en automates temporisés

Tissage

- Services abstraits en automates temporisés
 - sur approximation des traces d'exécution
 - sur approximation du comportement temporel
- Aspects traduits en automates temporisés
- Tissage par produit d'automates temporisés
 - sélectionne les traces autorisées
 - coupe ou allonge les traces interdites

	l_0	:	<code>getUser()</code>	\rightsquigarrow	l_1
	l_1	:	<code>M1.prendre()</code>	\rightsquigarrow	l'_1
	l'_1	:	<code>reset(i, 25)</code>	\rightsquigarrow	l_2
	l_2	:	<code>M2.prendre()</code>	\rightsquigarrow	l_3
$S1 \times A1 =$	l_3	:	<code>S1.calcul()</code>	\rightsquigarrow	l_4
	l_4	:	<code>M2.relacher()</code>	\rightsquigarrow	l_5
	l_5	:	<code>M1.relacher()</code>	\rightsquigarrow	l'_6
	l'_6	:	<code>cancel(i)</code>	\rightsquigarrow	l_6
	l_6	:	<code>endUser()</code>	\rightsquigarrow	l_0

Bilan

- Autres étapes
 - Prise en compte des durées d'exécution
 - fonction de coût $f_{cout}(a) = [min, max]$
 - évite d'introduire des minuteurs (ou
 - Vérification de propriétés de disponibilité
 - par ex. S1 traite une requête en moins de 45 sec.
 - utilisation d'UPPAAL
 - Concrétisation

Bilan

- Autres étapes
 - Prise en compte des durées d'exécution
 - fonction de coût $f_{cout}(a) = [min, max]$
 - évite d'introduire des minuteurs (ou
 - Vérification de propriétés de disponibilité
 - par ex. S1 traite une requête en moins de 45 sec.
 - utilisation d'UPPAAL
 - Concrétisation
- Généralisation de l'approche
 - aux propriétés et automates temporisés
 - insertion d'interruption et d'attente

Plan

- 1 Eléments de programmation par aspects
- 2 Application au contrôle d'accès
- 3 Application à la fusion de composants
- 4 Application à la disponibilité
- 5 Conclusion

Conclusion

- Approche commune
 - aspect : propriété de sûreté (ensemble de traces autorisées)
 - tissage : produit entre programme et aspect
 - modularité (séparation des problèmes)
 - formel et efficace

Conclusion

- Approche commune
 - aspect : propriété de sûreté (ensemble de traces autorisées)
 - tissage : produit entre programme et aspect
 - modularité (séparation des problèmes)
 - formel et efficace
- Politiques de contrôle d'accès
 - couper l'exécution si le programme viole la politique
 - insérer des aborts

Conclusion

- Approche commune
 - aspect : propriété de sûreté (ensemble de traces autorisées)
 - tissage : produit entre programme et aspect
 - modularité (séparation des problèmes)
 - formel et efficace
- Contraintes d'ordonnement
 - choisir les entrelacements satisfaisant les contraintes
 - séquentialiser l'exécution

Conclusion

- Approche commune
 - aspect : propriété de sûreté (ensemble de traces autorisées)
 - tissage : produit entre programme et aspect
 - modularité (séparation des problèmes)
 - formel et efficace
- Politiques de disponibilité
 - sélectionner des comportements temporels
 - insérer des `waits` ou des `aborts`

En cours et futur

- Combiner ordonnancement et disponibilité

En cours et futur

- Combiner ordonnancement et disponibilité
- Généralisation de la fusion à des réseaux non déterministes

En cours et futur

- Combiner ordonnancement et disponibilité
- Généralisation de la fusion à des réseaux non déterministes
- Aspects de tolérance aux fautes