
*Programmation par aspects
et
Tissage de propriétés de sécurité*

Pascal Fradet

INRIA (RENNES & RHÔNE-ALPES)

ACI DISPO - 18 novembre 2003

Plan

- DISPO et Aspects
- Éléments de programmation par aspects
 - ✓ L'idée, les choix, les problèmes
 - ✓ Vision statique (transformation de programme)
 - ✓ Vision dynamique (moniteur observant/modifiant les traces d'exécution)
- Imposer des propriétés de sûreté
- Directions de recherche

DISPO et Aspects

- Ce qui nous intéresse:

Imposer automatiquement des propriétés de disponibilité à des composants logiciel

- Ce que l'on a fait

Imposer automatiquement des propriétés de sûreté à des programmes

- D'où :

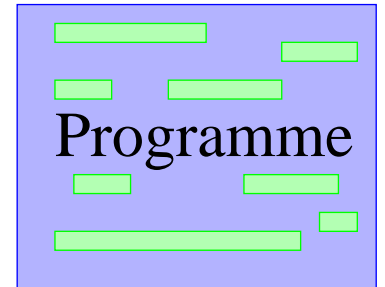
- ✓ modulariser les techniques
- ✓ passer de la sûreté à la disponibilité

Programmation par aspects

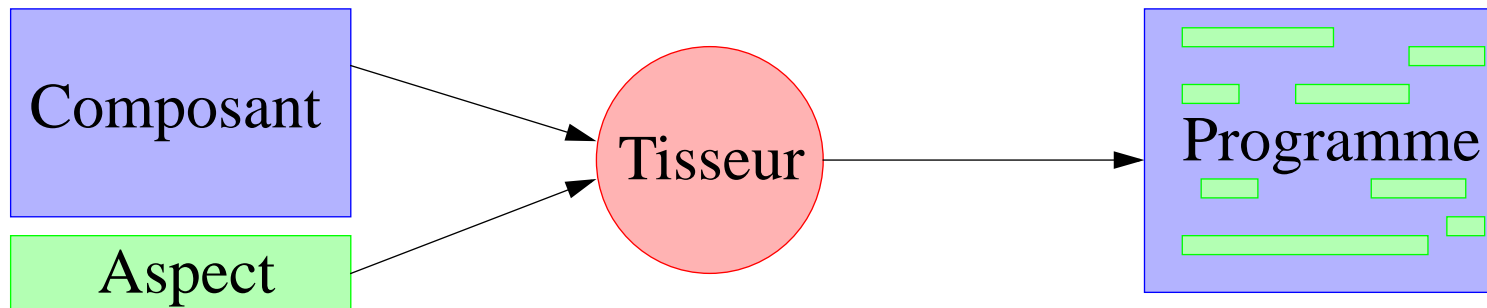
L'idée

- L'implémentation de certains aspects traverse le programme

- ✓ par ex. profilage, sécurité, synchronisation
- ✓ abstractions (fonctions, procédures, classes)
inadaptées



- Programmation par Aspects [G. Kiczales *et al.* 1997]



Programmation par aspects

Les choix

○ Nature des aspects

- ✓ propriétés fonctionnelles
- ✓ propriétés non fonctionnelles
- ✓ optimisations

○ Langage d'aspects

- ✓ langage général *vs.* langage dédié
- ✓ voit la syntaxe et/ou la sémantique du composant
- ✓ directives d'insertion de code *vs.* propriétés à imposer

○ Tisseur

- ✓ statique (transformation de programme)
- ✓ dynamique (moniteur)

Programmation par aspects

Problèmes

- Interactions entre aspects, débogage, réutilisation, ...
- Important de pouvoir
 - ✓ raisonner sur les programmes (i.e. sans regarder le tissage)
 - ✓ maîtriser l'impact sémantique
- Paradigme sans modèle
 - ✓ Très expressif et peu de garde-fous
 - ✓ Problèmes exacerbés lors de la composition de plusieurs aspects

Vision statique

- On voit le composant de base comme un arbre de syntaxe abstraite (AST)
- Aspect : transformations d'AST (*motif* \rightarrow *terme*)
 - ✓ Sur la syntaxe : motifs sur les opérateurs, les noms de variables/fonctions
 - ✓ Sur la sémantique : motifs sur l'état (valeurs de variables, ...)
- Version simplifiée:
 - ✓ aspect = directives d'insertion de code
 - ✓ tisseur = une passe sur le code

AspectJ

Kiczales *et al.*

○ Langage d'aspect pour Java

- ✓ général
- ✓ voit la syntaxe et la sémantique du composant
- ✓ directives d'insertion de code
- ✓ tissage statique (instrumentation de code)

○ Trois constituants

- ✓ points de jointure (*join points*)
 - ce qu'il est possible d'observer
- ✓ motifs (*pointcuts*)
 - sélectionne certains points de jointure
- ✓ inserts (*advice*)
 - code à exécuter aux points de jointure sélectionnés

AspectJ

Un exemple simple d'aspect

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

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

Vision dynamique

- On voit le composant de base comme ses *traces d'exécution*
- Aspect : insertion de code à certains points de l'exécution
 - ✓ syntaxique : motifs sur les opérateurs, les noms de variables/fonctions, ...
 - ✓ sémantique : motifs sur la valeurs de variables, l'état de la pile d'appel, ...
- Choix :
 - ✓ points de jointure : *stateless* ou *stateful*
 - ✓ inserts : visibles ou invisibles
- Tisseur : moniteur observant/modifiant l'exécution
 - ✓ expression sur la sémantique
 - ✓ Mise en oeuvre statique plus compliquée
 - ✓ Contrôle de l'impact sémantique plus simple

Tissage de propriétés

Travail commun avec T. Colcombet, IRISA, Rennes

○ Motivations:

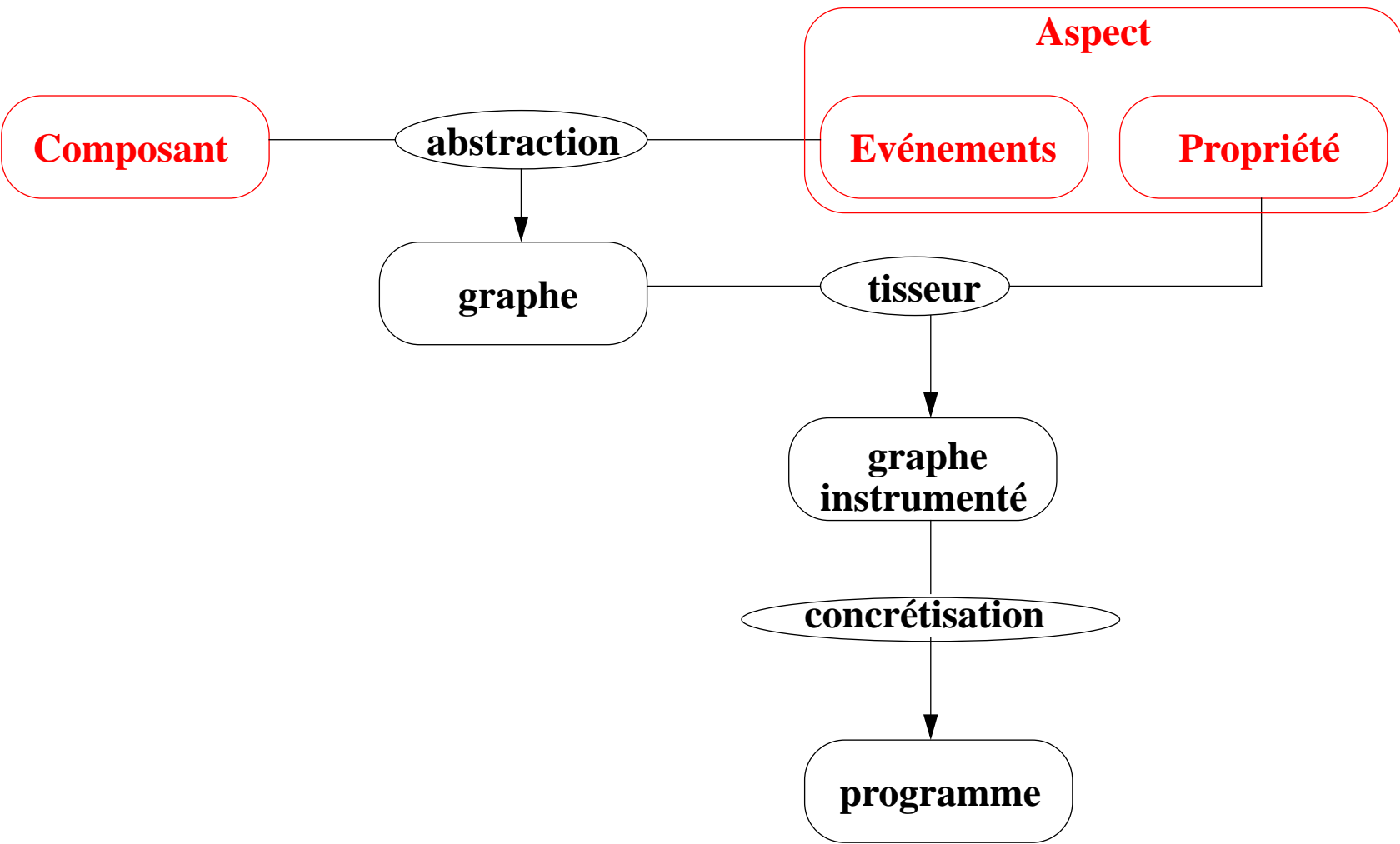
- ✓ Contrôler l'impact des aspects
- ✓ Imposer des propriétés de sûreté
 - par ex: politiques de sécurité qui sont difficiles à implémenter et à vérifier

○ Cadre

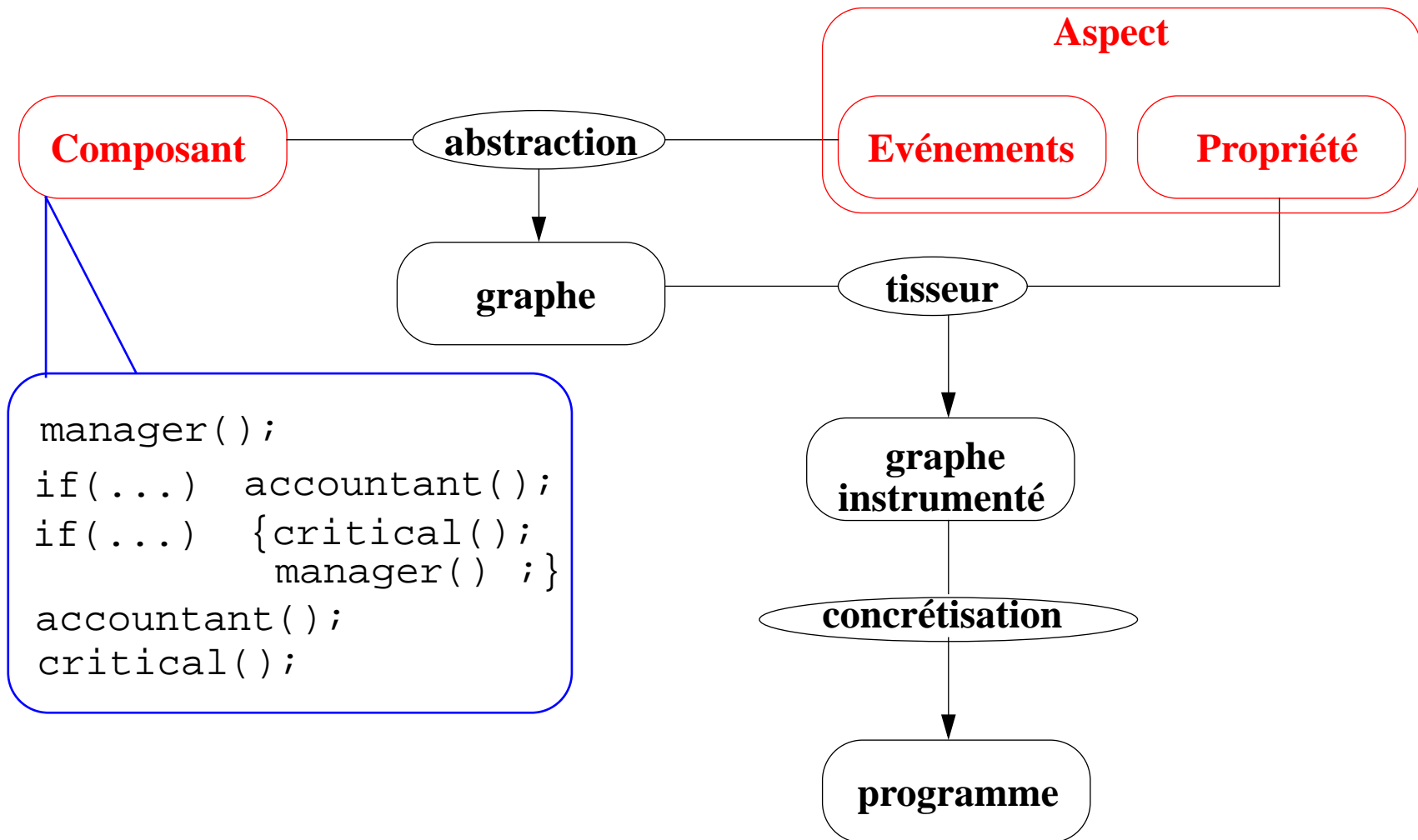
- ✓ inserts = skip ou abort
- ✓ tissage statique

○ Défi: efficacité du tissage

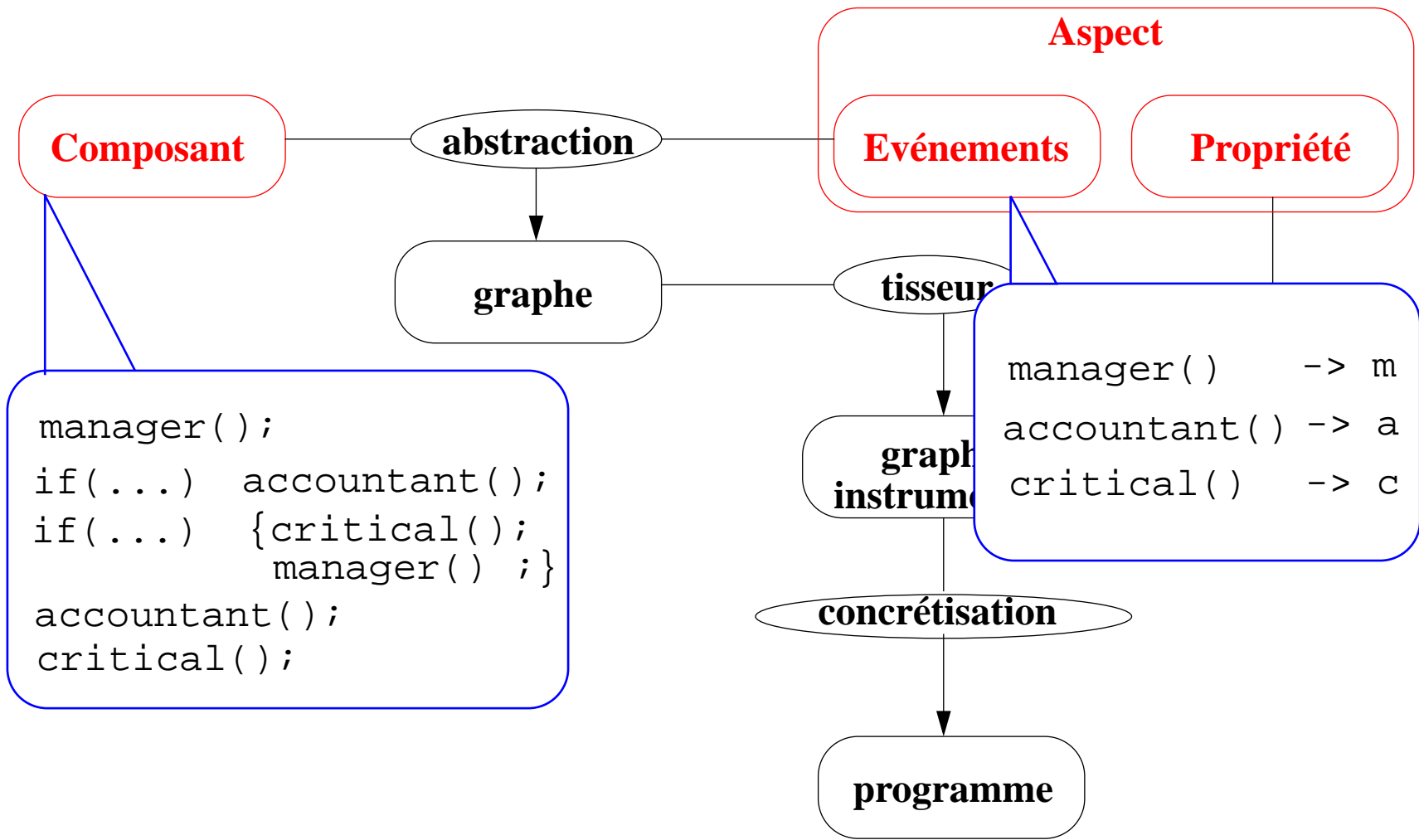
Survol



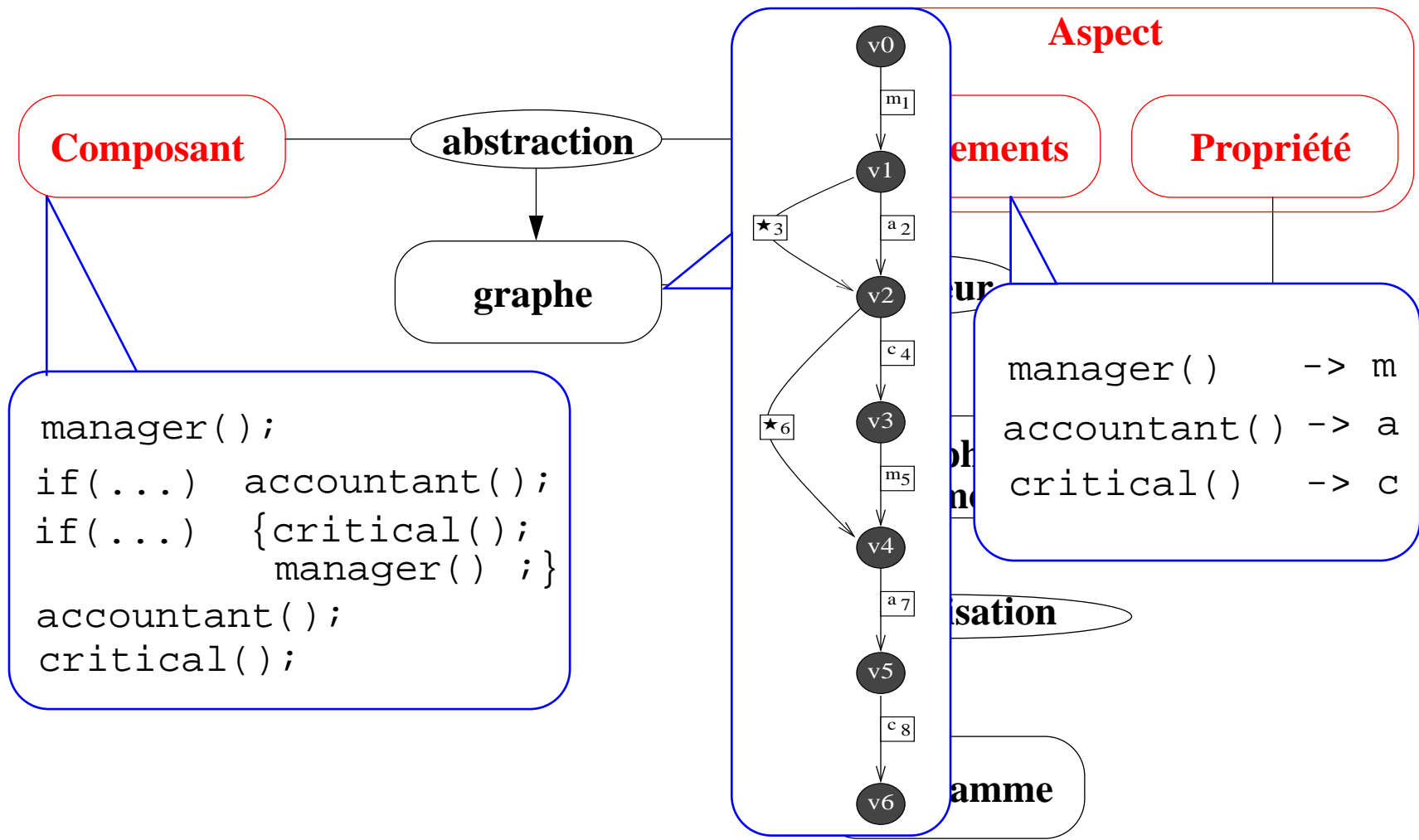
Un exemple (1)



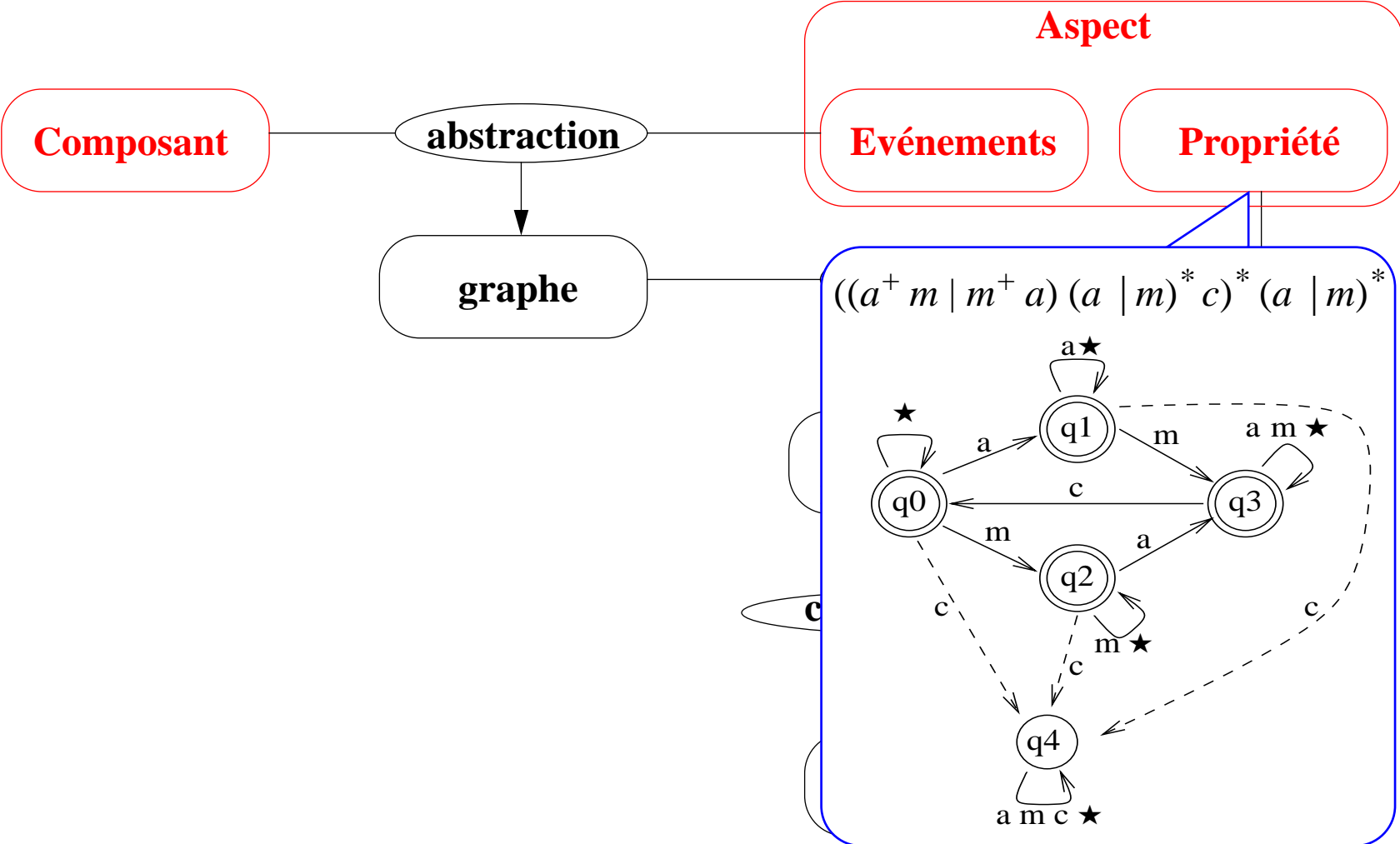
Un exemple (2)



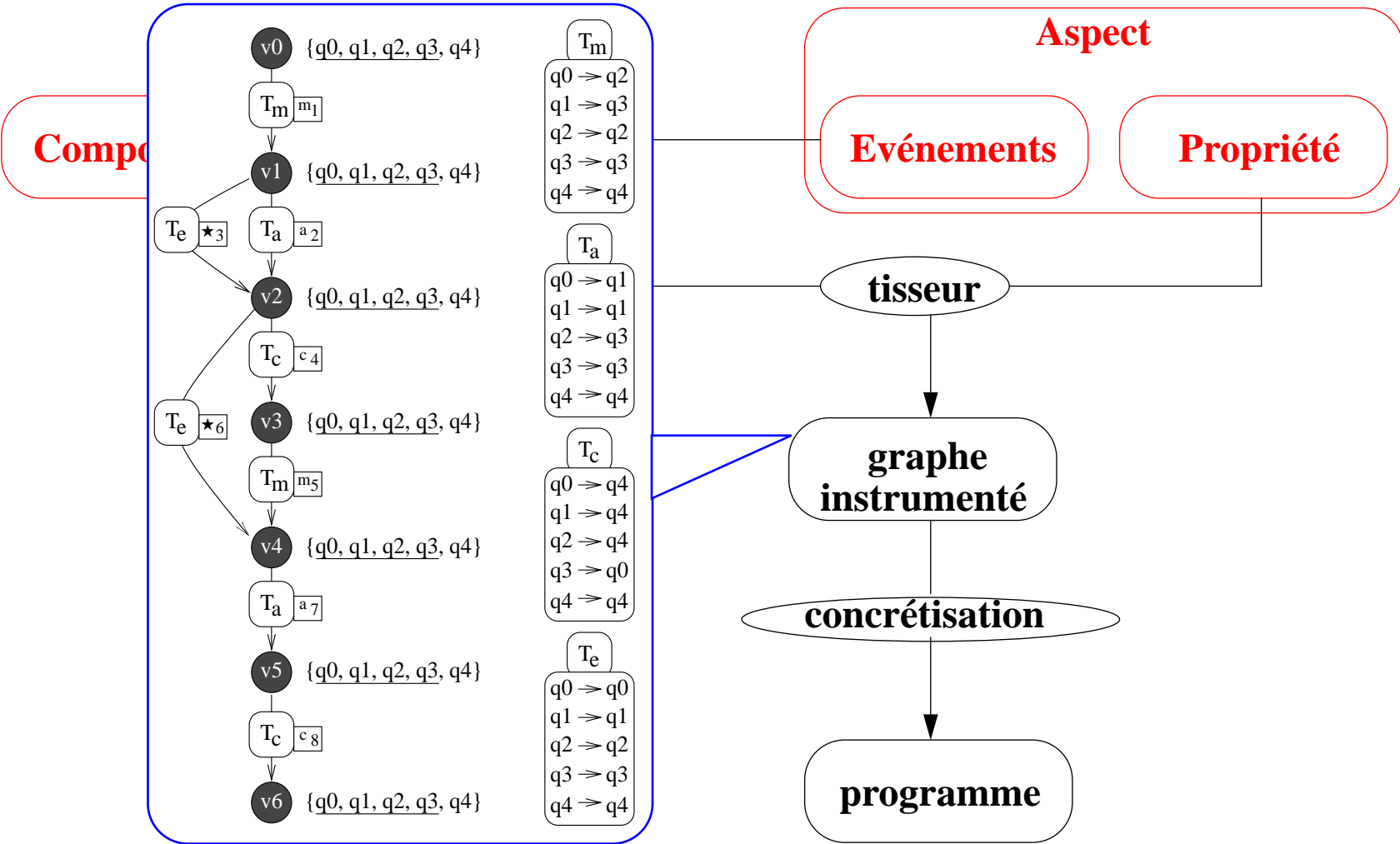
Un exemple (3)



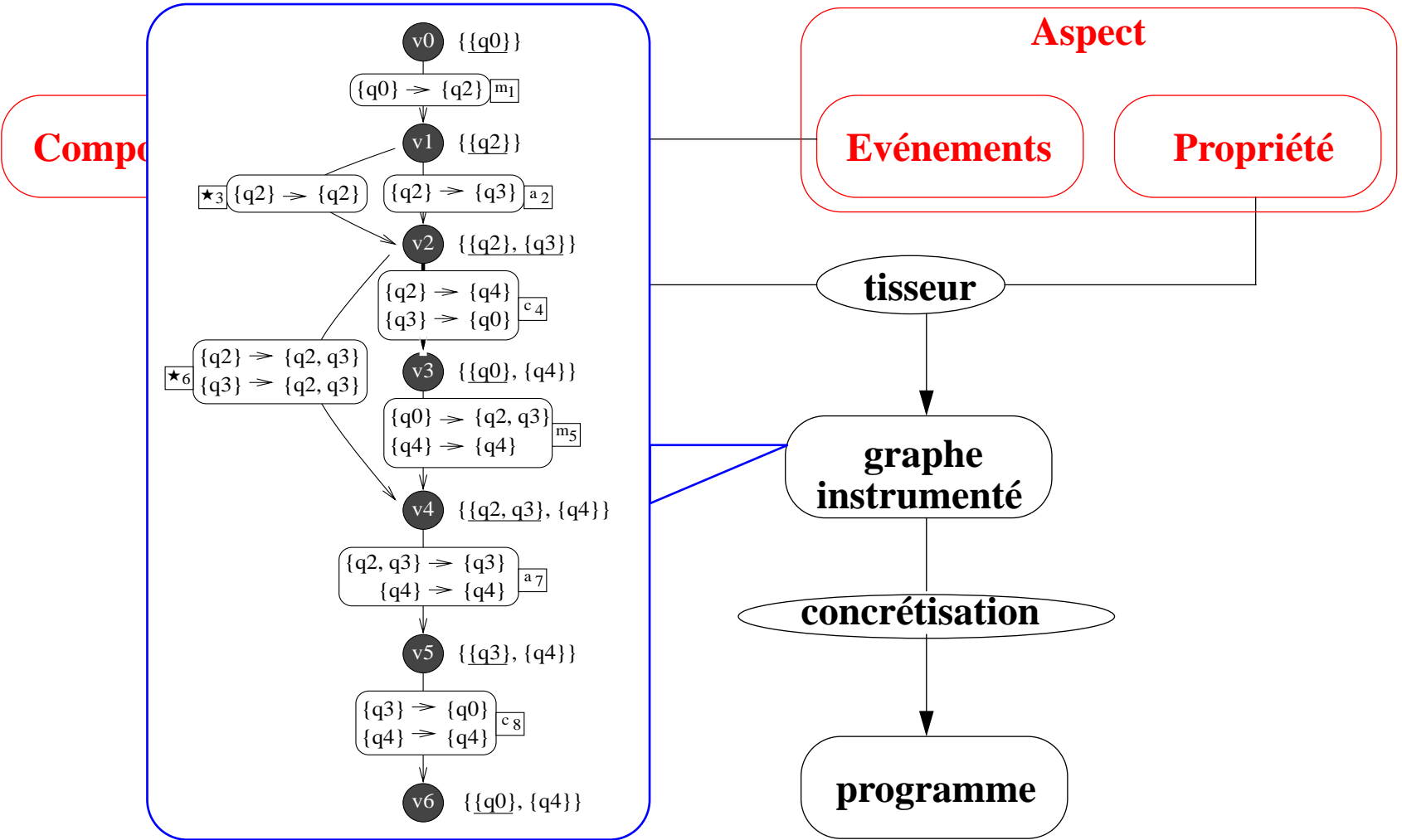
Un exemple (4)



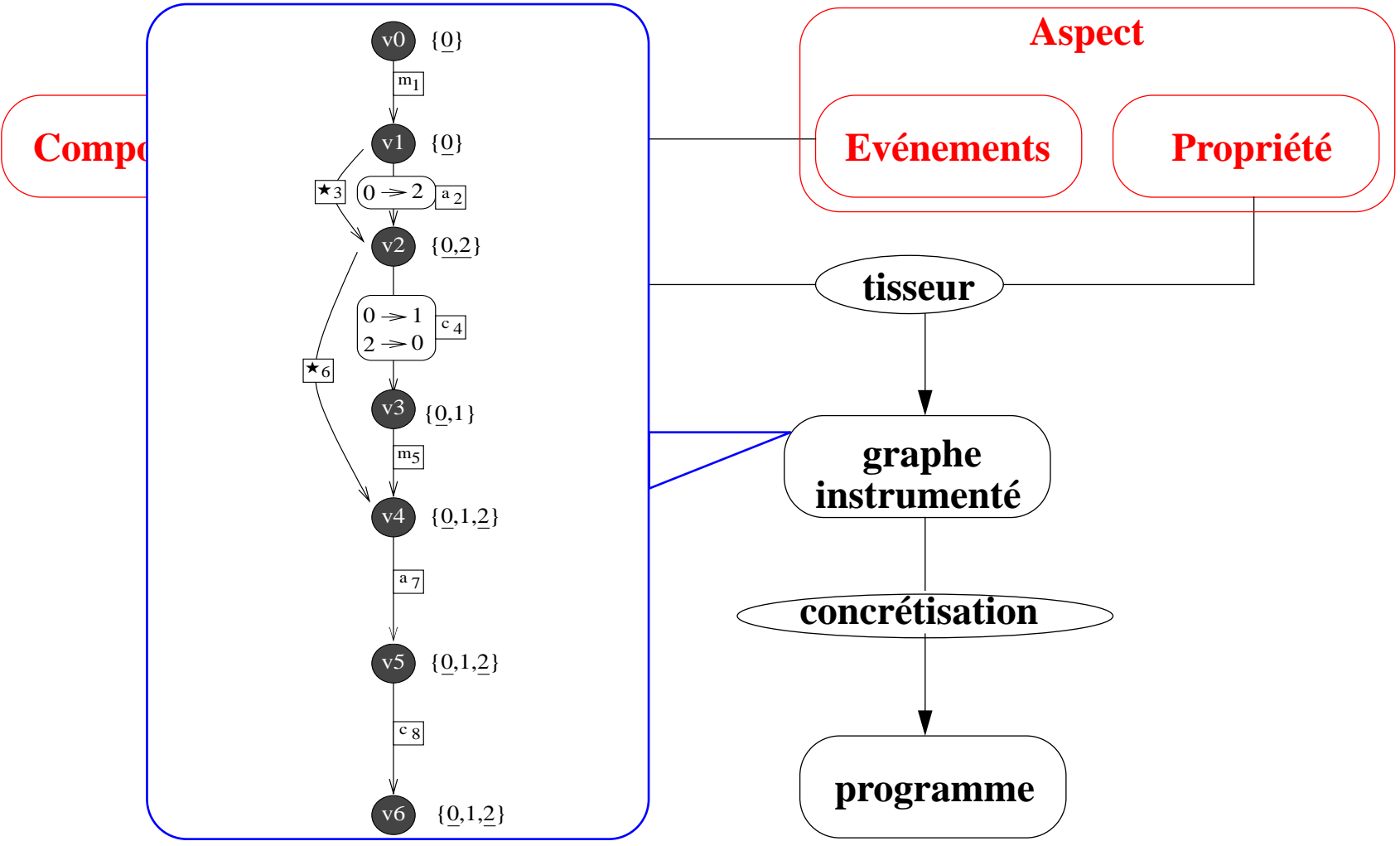
Un exemple (5)



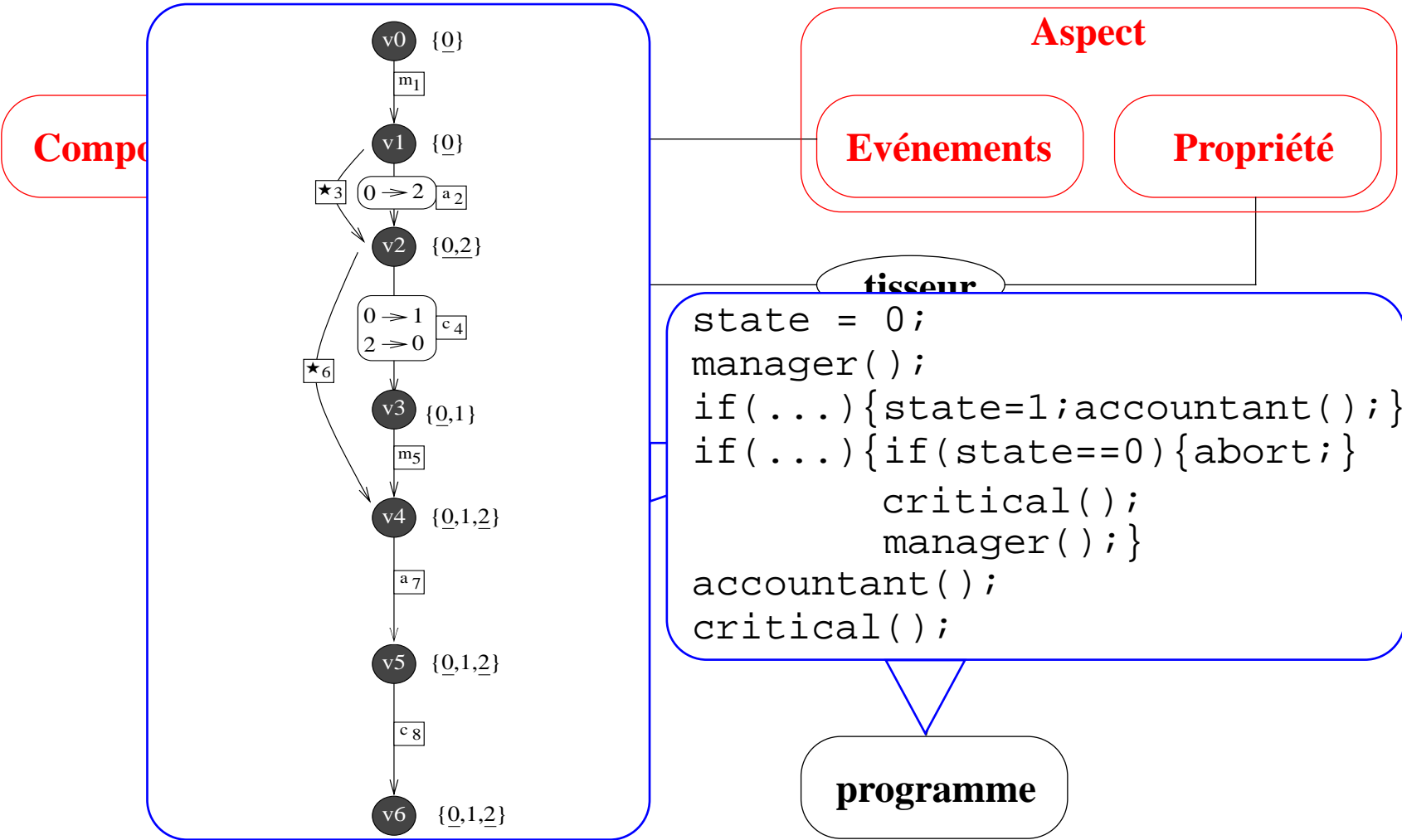
Un exemple (6)



Un exemple (7)



Un exemple (8)



Extensions

○ Propriétés sur événements sémantiques

- ✓ transformation du code
- ✓ ex: `x=e` remplacé par `if e==0 then x=0 else x=e`
- ✓ analyse statique

○ Graphe de flot de contrôle inter-procéduraux

- ✓ traces (avec `call` et `return`) plus précises

○ Graphes d'appels

- ✓ trace = pile d'appels
- ✓ permet d'exprimer le modèle de sécurité de Java (inspection de pile)

Bilan

- Application à la sécurisation de code mobile
 - ✓ composant = applet téléchargée
 - ✓ aspect = politique de sécurité locale
 - ✓ tissage à la réception

- Avantages de l'approche
 - ✓ efficacité et modularité (maintenance)
 - ✓ pas de rejet de programmes
 - ✓ sémantique claire

Directions de recherche

○ De la sûreté à la disponibilité

- ✓ de la disponibilité à la sûreté
- ✓ analyses de programme (pour éviter des propriétés de sûreté trop fortes)

○ Aspects et composants

- ✓ modularisation du tissage (analyses et transformations)
- ✓ tissage de réseau (composants opaques)

○ Un début : assemblage et fusion de composants

- ✓ modèle simple, formel et expressif
- ✓ permet d'imposer des propriétés pendant la fusion