Visiteur

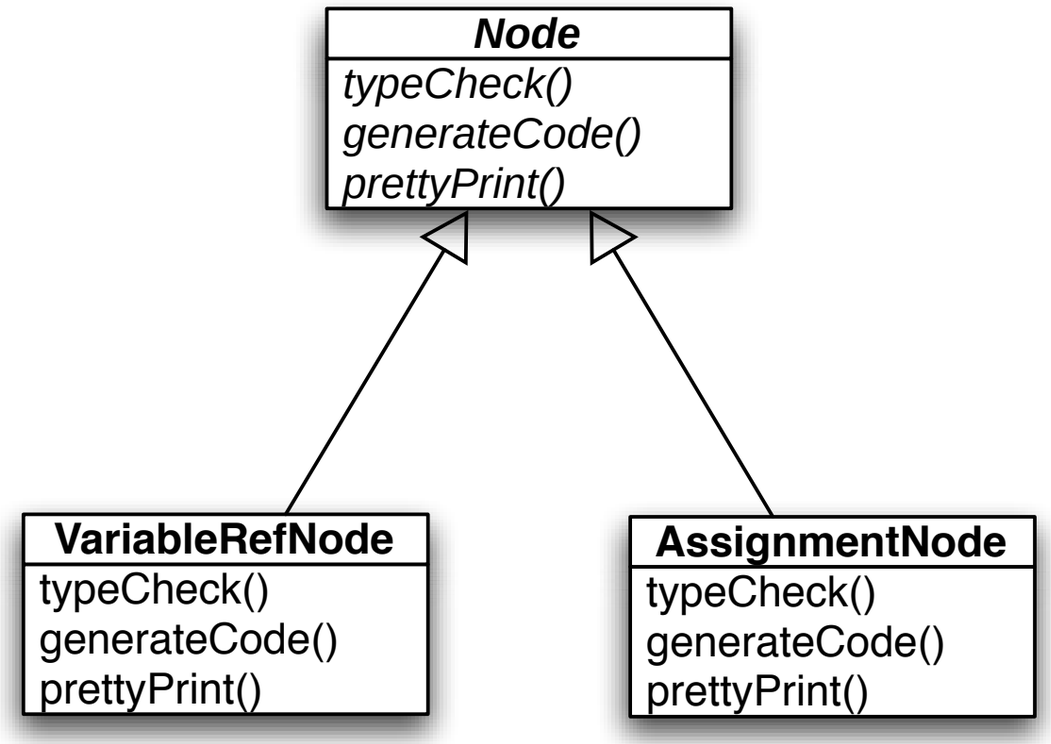
Visitor

Objectif

- Représenter une opération qui sera exécutée sur une structure d'objets. Le visiteur permet la définition d'une nouvelle opération sans changer les classes sur lesquelles elle s'exécute

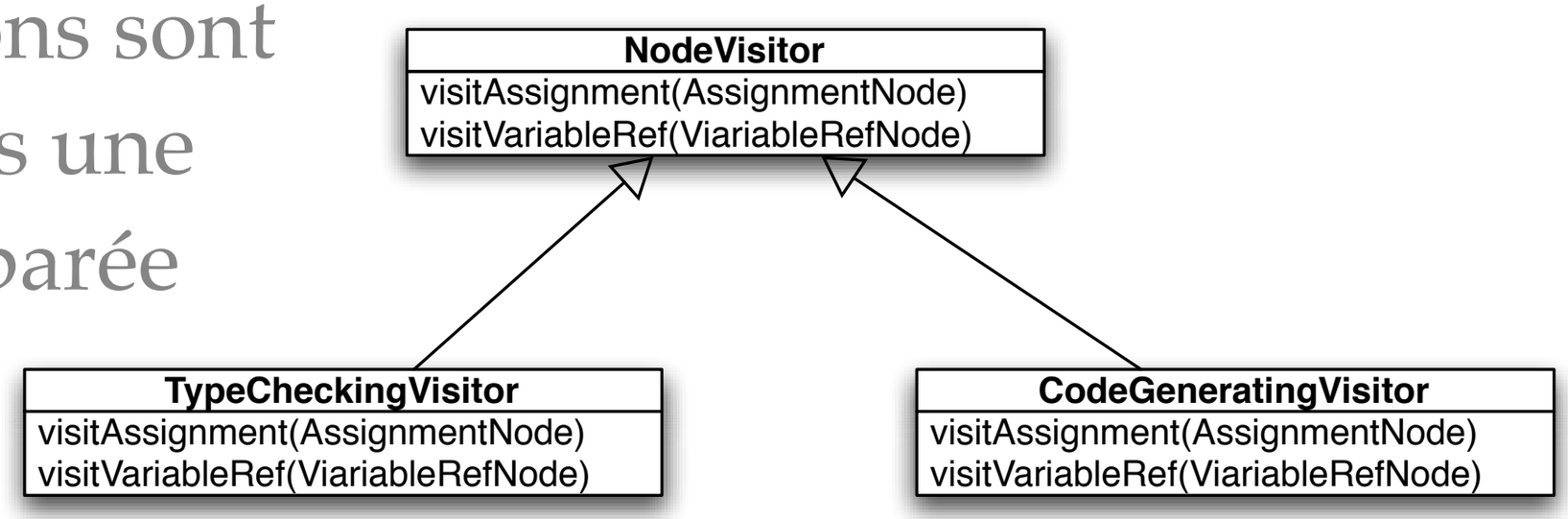
Motivation (1/3)

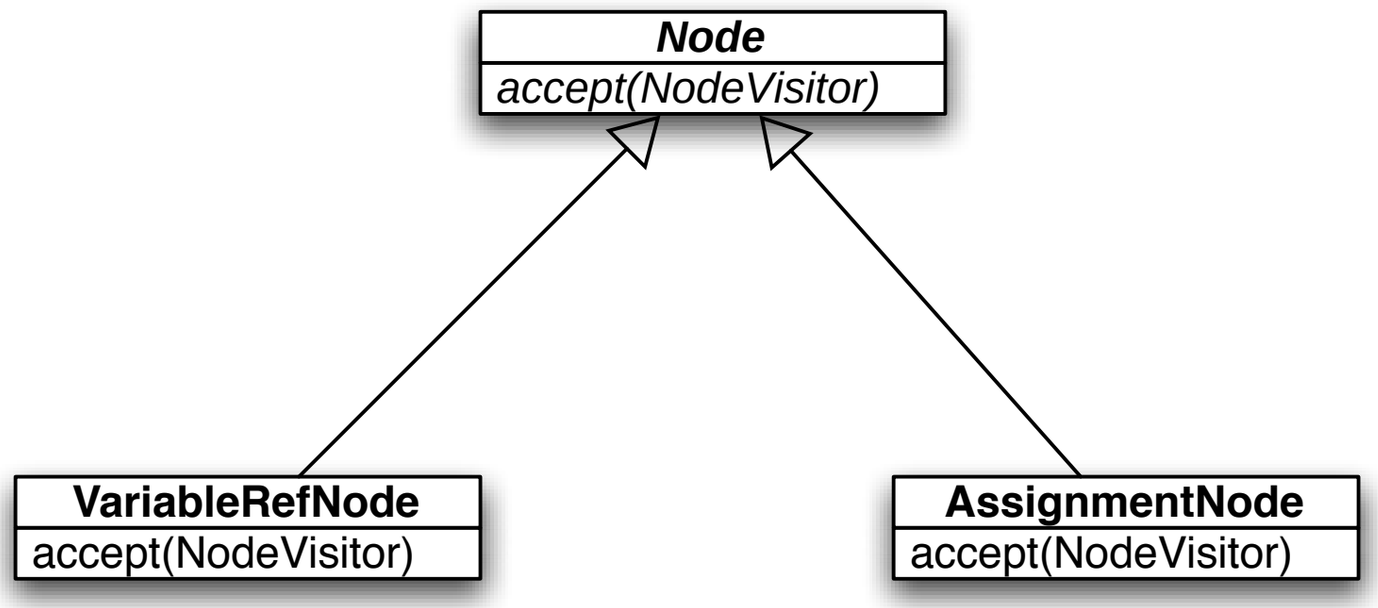
- Exemple: un compilateur
- Différentes opérations peuvent s'appliquer a cette structure: vérification de type, optimisation, etc.



Motivation (2/3)

- Ces opérations sont définies dans une structure séparée

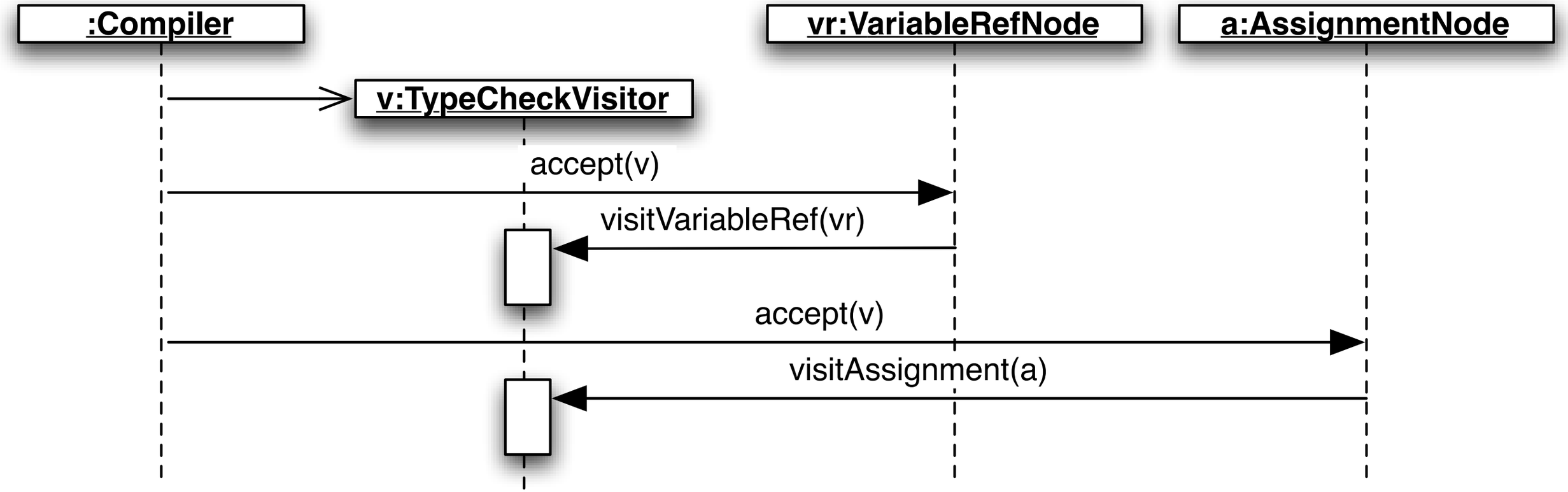





```
VariableRefNode::accept(v : NodeVisitor) {
    v.visitVariableRef(self)
}
```

```
AssignmentNode::accept(v : NodeVisitor) {
    v.visitAssignment(self)
}
```

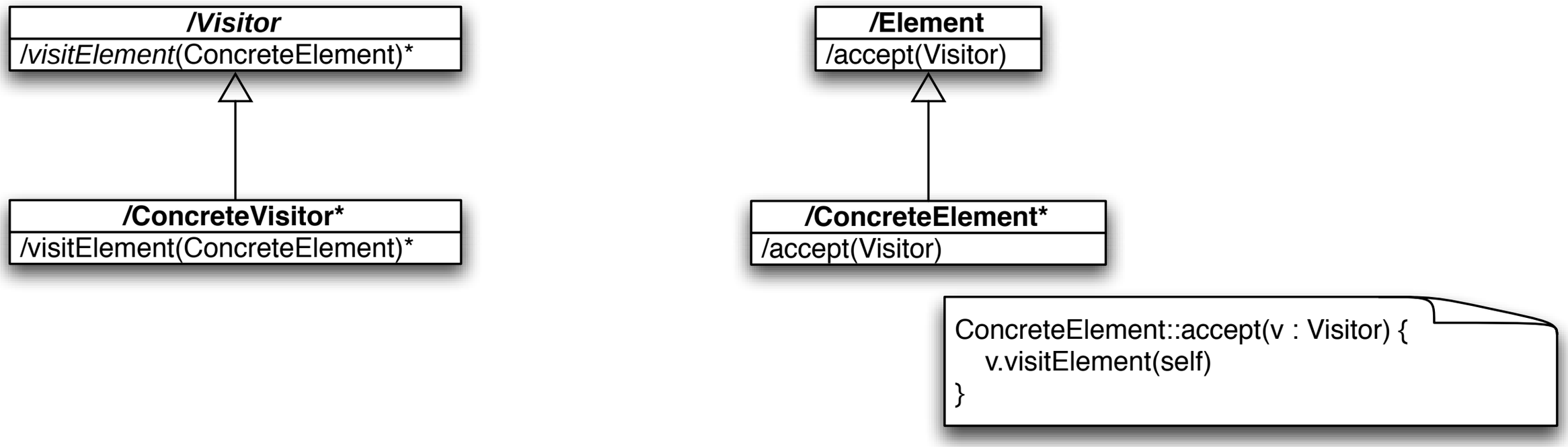
Motivation (3/3)



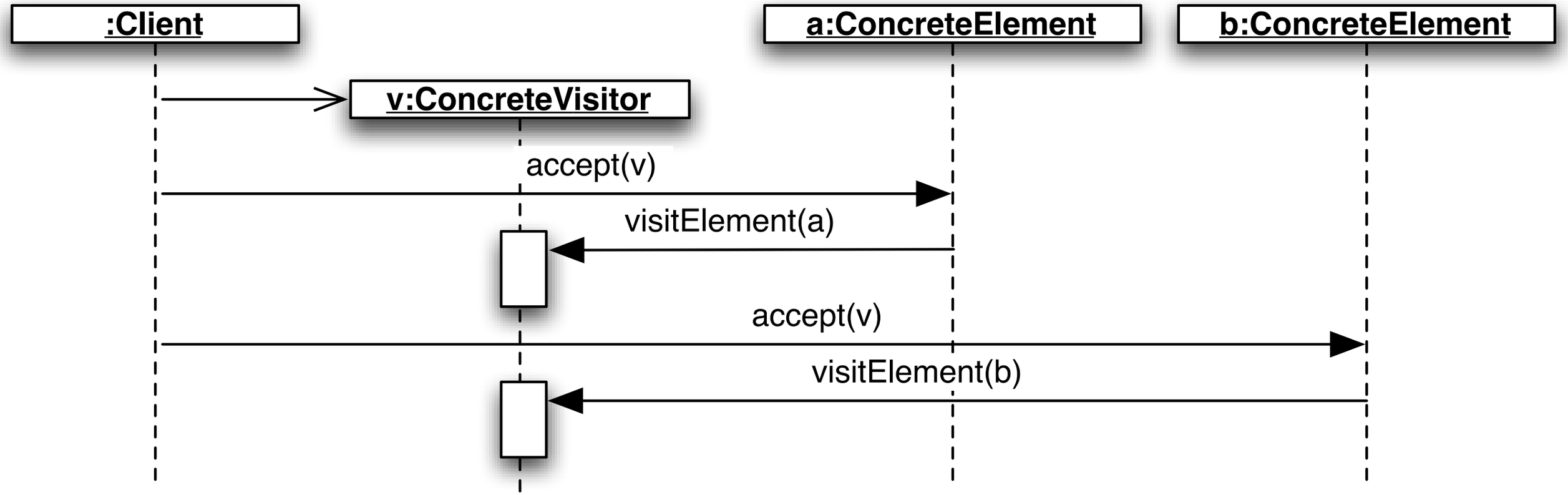
Applicabilité

- Utiliser le Visiteur quand:
  - on a besoin d'effectuer une opération sur une structure d'objets appartenant à des classes différentes
  - plusieurs opérations hétérogènes s'appliquent à ces objets, et on ne souhaite pas "polluer" les classes avec ces opérations
  - la structure ne change pas

Structure


/Visitor
/visitElement(ConcreteElement)*
/ConcreteVisitor*
/visitElement(ConcreteElement)*
/Element
/accept(Visitor)
/ConcreteElement*
/accept(Visitor)
ConcreteElement::accept(v : Visitor) {
v.visitElement(self)
}

Collaborations



Conséquences

- L'ajout de nouvelles opérations est simplifié
- Le visiteur groupe les opérations en relation et les isole des autres
- L'ajout d'une nouvelle classe de la structure devient complexe
- Accumulation de valeurs
- L'encapsulation est violée

Compromis d'implémentation

- *Double dispatch* (CLOS, Dylan) : l'opération exécutée dépend du fournisseur et du client
- Qui est responsable de la traversée? La structure, le visiteur ou un Itérateur?

Visiteur

Visitor