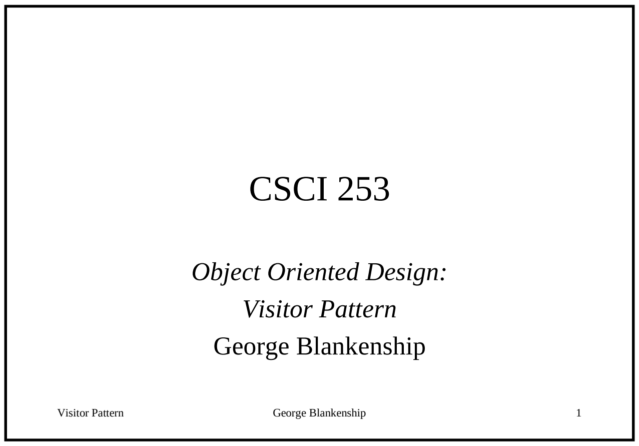
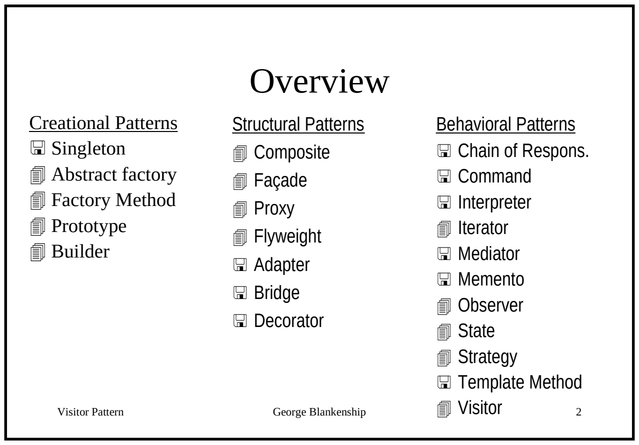


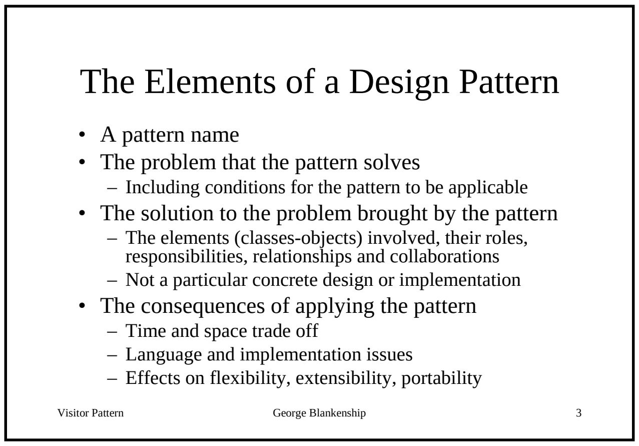
CSCI 253

Object Oriented Design:
Visitor Pattern
George Blankenship



Overview

Creational Patterns	Structural Patterns	Behavioral Patterns
🖫 Singleton	🗐 Composite	🖫 Chain of Respons.
🗐 Abstract factory	🗐 Façade	🖫 Command
🗐 Factory Method	🗐 Proxy	🖫 Interpreter
🗐 Prototype	🗐 Flyweight	🗐 Iterator
🗐 Builder	🖫 Adapter	🖫 Mediator
	🖫 Bridge	🖫 Memento
	🖫 Decorator	🗐 Observer
		🗐 State
		🗐 Strategy
		🖫 Template Method
		🗐 Visitor



The Elements of a Design Pattern

- A pattern name
- The problem that the pattern solves
  - Including conditions for the pattern to be applicable
- The solution to the problem brought by the pattern
  - The elements (classes-objects) involved, their roles, responsibilities, relationships and collaborations
  - Not a particular concrete design or implementation
- The consequences of applying the pattern
  - Time and space trade off
  - Language and implementation issues
  - Effects on flexibility, extensibility, portability

The Visitor Pattern: The Problem

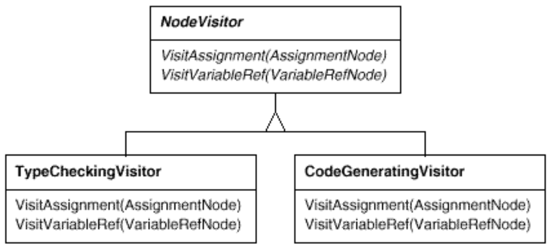
Represents an operation to be performed on the elements of an object structure. Visitor lets you define a new operation without changing the classes of the elements on which it operates

• many distinct and unrelated operations need to be performed on objects in an object structure an you want to avoid "polluting" their classes with these operations
• the classes defining the object structure rarely change but you often want to define new operations over the structure

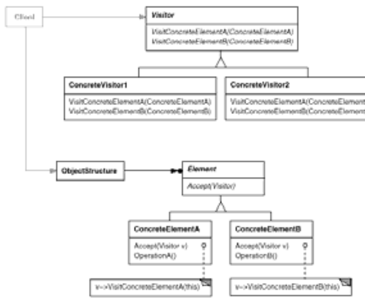
Visitor Example

- The root of the structure accepts a visitor:
  root.accept(visitor)
- The root and every child object in the structure have a an accept method:
```
void accept( visitor ) {
        visitor.visit( this );
        for each child of mine
                child.accept( visitor ) next
}
```
- In short order our visitor gets a visit() call from each object in the collection.
- An interesting collection might contain different types of objects. To handle this, the visitor overloads the method visit() with a variety of argument types.
- The visitor interface for an HTML tool has methods
```
        visit( Document node )
        visit( Tag node )
        visit( TagBlock node )
        visit( Comment node )
        visit( Text node )
```
- If a Document object calls visit(this), the first method is invoked. If a Comment object calls visit(this), the fourth method is invoked.

NodeVisitor


NodeVisitor
VisitAssignment(AssignmentNode)
VisitVariableRef(VariableRefNode)
TypeCheckingVisitor
VisitAssignment(AssignmentNode)
VisitVariableRef(VariableRefNode)
CodeGeneratingVisitor
VisitAssignment(AssignmentNode)
VisitVariableRef(VariableRefNode)

The Vistor Pattern: Structure


Client
Visitor
VisitConcreteElementA(ConcreteElementA)
VisitConcreteElementB(ConcreteElementB)
ConcreteVisitor1
VisitConcreteElementA(ConcreteElementA)
VisitConcreteElementB(ConcreteElementB)
ConcreteVisitor2
VisitConcreteElementA(ConcreteElementA)
VisitConcreteElementB(ConcreteElementB)
ObjectStructure
Element
Accept(Visitor)
ConcreteElementA
Accept(Visitor v)
OperationA()
ConcreteElementB
Accept(Visitor v)
OperationB()
v->VisitConcreteElementA(this)
v->VisitConcreteElementB(this)

Visitor Pattern George Blankenship 7

The Visitor Pattern: Participants

- Context
  - Declares a Visit operation for each class of ConcreteElement in the object structure
  - The operations name and signature identified the class that sends the Visit request
- ConcreteVisitor
  - Implements each operation declared by Visitor
  - Each operation implements a fragment of the algorithm for the corresponding class of object in the object structure
  - Provides the context for the algorithm and stores its state (often accumulating results during traversal)
- Element
  - Defines an accept operation that takes a visitor as an argument

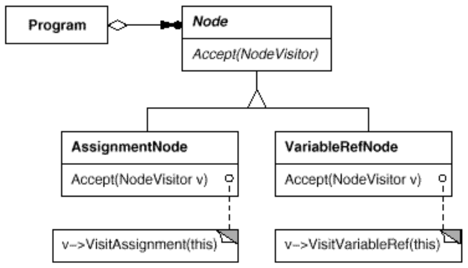
Visitor Pattern George Blankenship 8

The Visitor Pattern: Collaboration

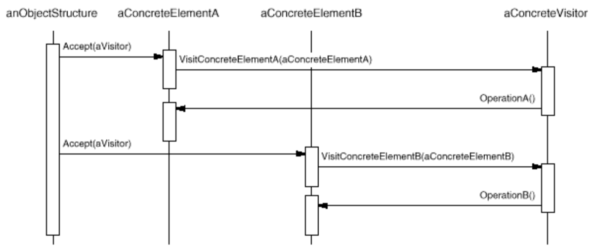
- ConcreteElement
  - Implements an accept operation that takes a visitor as an argument
- ObjectStructure
  - Can enumerate its elements
  - May provide a high-level interface to allow the visitor to visit its elements
  - May either be a Composite or a collection such as a list or a set
- A client that uses the visitor pattern must create a ConcreteVisitor object and then traverse the object structure visiting each element with the visitor
- When an element is visited, it calls the Visitor operation that corresponds to its class. The element supplies itself as an argument to this operation

Visitor Pattern George Blankenship 9

Participant Map



Process Flow



The Visitor Pattern: Consequences

- Makes adding new operations easy: a new operation is defined by adding a new visitor (in contrast, when you spread functionality over many classes each class must be changed to define the new operation)
- Gathers related operations and separates unrelated ones: related behavior is localised in the visitor and not spread over the classes defining the object structure
- Adding new ConcreteElement classes is hard:each new ConcreteElement gives rise to a new abstract operation in Visitor and a corresponding implementation in each ConcreteVisitor

The Visitor Pattern: Visiting

- Allows visiting accross class hierarchies: an iterator can also visit the elements of an object structure as it traverses them and calls operations on them but all elements of the object structure then need to have a common parent. Visitor does not have this restriction.
- Accumulating state: visitor can accumulate state as it proceeds with the traversal. Without a visitor this state must be passed as an extra parameter of handled in global variables
- Breaking encapsulation :Visitor's approach assumes that the ConcreteElement interface is powerful enough to allow the visitors to do their job. As a result the pattern ofthen forces to provide public operations that access an element's internal state which may compromise its encapsulation

The Visitor Pattern: Warnings

- An obvious problem is that the arguments and the return type of visiting methods have to be known in advance. A new Visitor class has to be defined, as well as a new accept method in every class of the hierarchy.
- The code is may be obscure.
- A lot of code has to be written to prepare the use of visitors: the visitor class with one abstract method per class, and a accept method per class. This code is tedious and boring to write. If we add a new class, the visitor class needs a new method. Furthermore, it is indeed likely that a new visiting method will need the definition of a new visitor pattern. At the least, several patterns have often to be written.
- If a visitor pattern has not been written in the first time, the hierarchy has to be modified to implement it. In particular, if the hierarchy cannot be modified because you are not allowed to, the visitor pattern cannot be applied at all.

The Visitor Pattern Implementation

- Double Dispatch. The key to visitor is a double dispatch: the meaning of the accept operation depends on the visitor and on the element. Languages that support double dispatch (CLOS) can do without this pattern.
- Who is responsible for traversing the object structure? Responsibility for traversal can be with:
  - The object structure
  - The visitor: is advisable when a particular complex traversal is needed (for example one that depends on the outcome of the operation), otherwise it I not advisable because a lot of traversal code will be duplicated in each ConcreteVisitor for each aggregate ConcreteElement
  - A separate iterator object