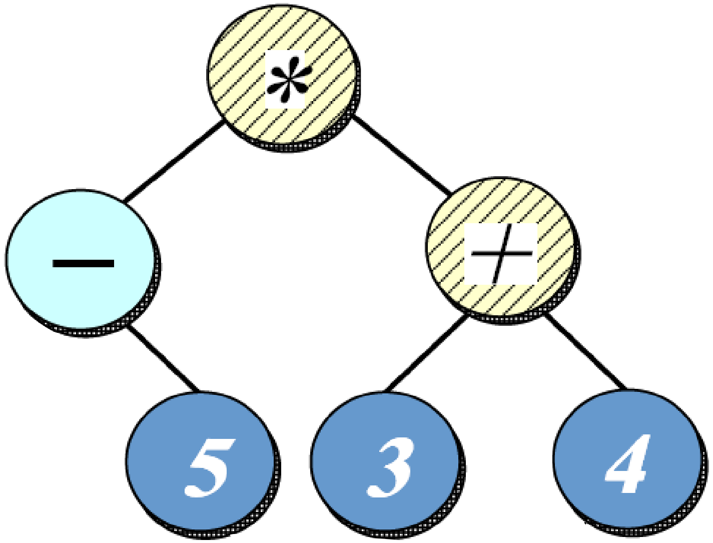
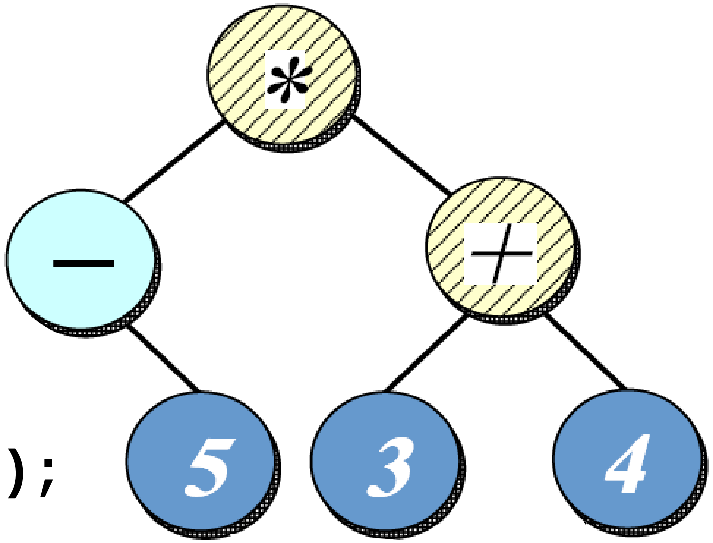
Solution (Part B): Decouple Operations from Expression Tree Structure

Visitor

- Defines action(s) at each step of traversal & avoids hard-coding action(s) into nodes



Solution (Part B): Decouple Operations from Expression Tree Structure



Visitor

- Defines action(s) at each step of traversal & avoids hard-coding action(s) into nodes
- Iterator calls **`accept(ET_Visitor&)`** method on each node in expression tree

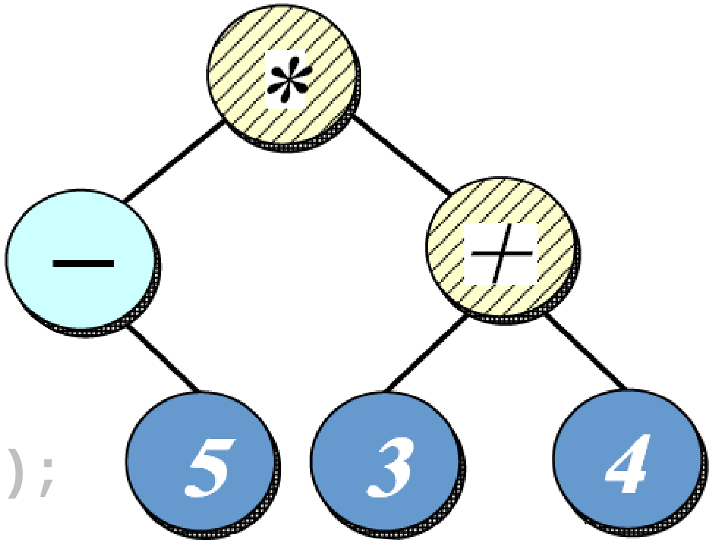
```
for (auto iter = expr_tree.begin();
     iter != expr_tree.end();
     ++iter)
  (*iter).accept(print_visitor);
```

Solution (Part B): Decouple Operations from Expression Tree Structure

Visitor

- Defines action(s) at each step of traversal & avoids hard-coding action(s) into nodes
- Iterator calls **`accept(ET_Visitor&)`** method on each node in expression tree

```
for (auto iter = expr_tree.begin();
          iter != expr_tree.end();
          ++iter)
   (*iter).accept(print_visitor);
```



- **`accept()`** calls back on visitor, e.g.:

```
void Leaf_Node::accept(ET_Visitor &v) {
  v.visit(*this);
}
```

Note “static polymorphism” based on method overloading by type

ET_Visitor Class Interface

- Interface for a visitor that defines operations performed for each type of node in the expression tree

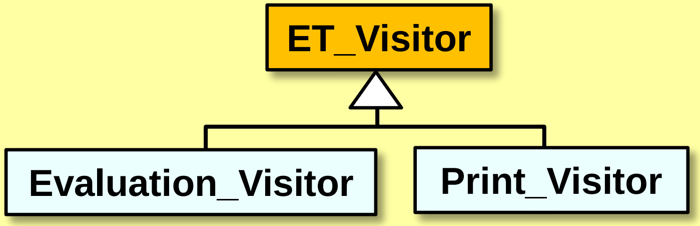
Interface

An overloaded visit() method is defined for each Component_Node subclass

```
virtual void visit(const Leaf_Node &node)=0
virtual void visit(const Composite_Negate_Node &node)=0
virtual void visit(const Composite_Add_Node &node)=0
virtual void visit(const Composite_Subtract_Node &node)=0
virtual void visit(const Composite_Divide_Node &node)=0
virtual void visit(const Composite_Multiply_Node &node)=0
```

- **Commonality**: Provides a common **`accept()`** method for all expression tree nodes & common **`visit()`** method for all visitor subclasses

- **Variability**: Can be subclassed to define specific behaviors for the visitors & nodes



Visitor GoF Object Behavioral

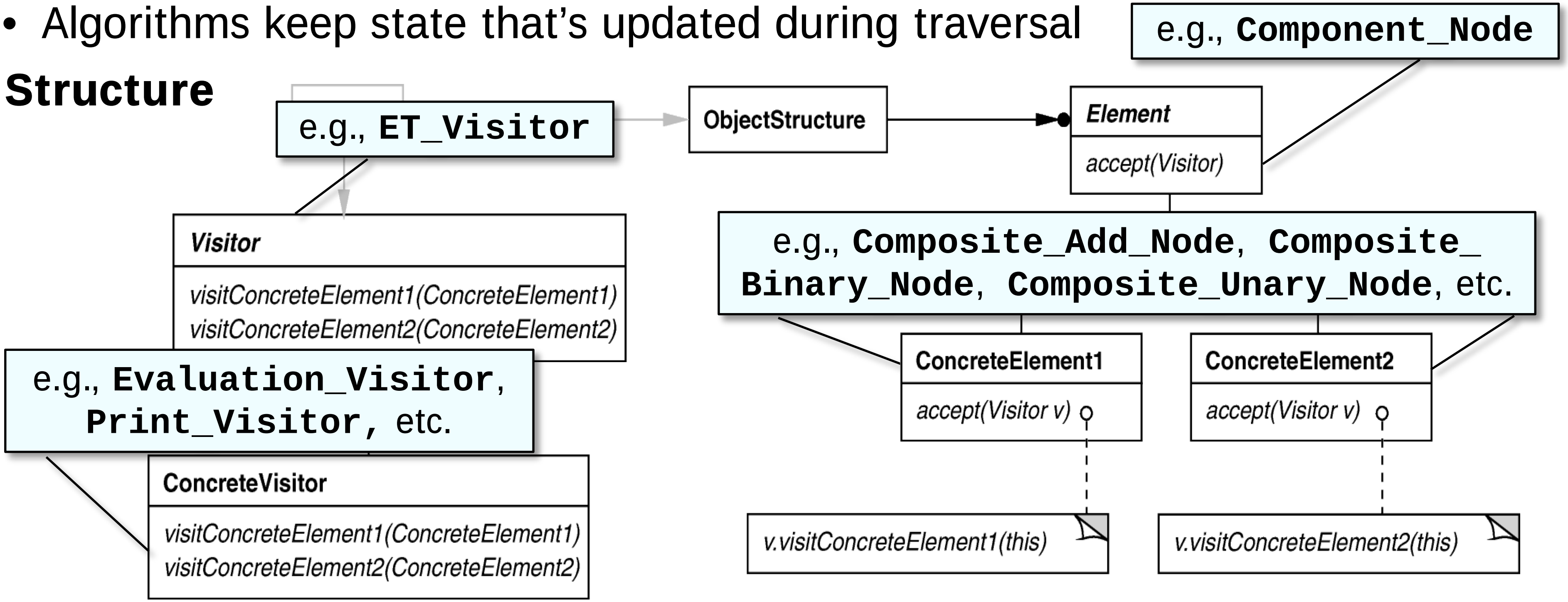
Intent

- Centralize operations on an object structure so that they can vary independently, but still behave polymorphically

Applicability

- When classes define many unrelated operations
- Class relationships in structure rarely change, but operations on them change
- Algorithms keep state that's updated during traversal

Structure


e.g., Component_Node
e.g., ET_Visitor
ObjectStructure
Element
accept(Visitor)
Visitor
visitConcreteElement1(ConcreteElement1)
visitConcreteElement2(ConcreteElement2)
e.g., Composite_Add_Node, Composite_Binary_Node, Composite_Unary_Node, etc.
ConcreteElement1
accept(Visitor v)
ConcreteElement2
accept(Visitor v)
e.g., Evaluation_Visitor, Print_Visitor, etc.
ConcreteVisitor
visitConcreteElement1(ConcreteElement1)
visitConcreteElement2(ConcreteElement2)
v.visitConcreteElement1(this)
v.visitConcreteElement2(this)

Visitor GoF Object Behavioral

Visitor implementation in C++

- The Print_Visitor class prints character code or value for each node

```
class Print_Visitor : public ET_Visitor {
public:
    virtual void visit(const Leaf_Node &);
    virtual void visit(const Add_Node &);
    virtual void visit(const Divide_Node &);
    // etc.
};
```

for all relevant Component_Node subclasses

Visitor GoF Object Behavioral

Visitor implementation in C++

- The Print_Visitor class prints character code or value for each node

```
class Print_Visitor : public ET_Visitor {
public:
    virtual void visit(const Leaf_Node &);
    virtual void visit(const Add_Node &);
    virtual void visit(const Divide_Node &);
    // etc.
};
```

- Can be combined with any traversal algorithm, e.g.:

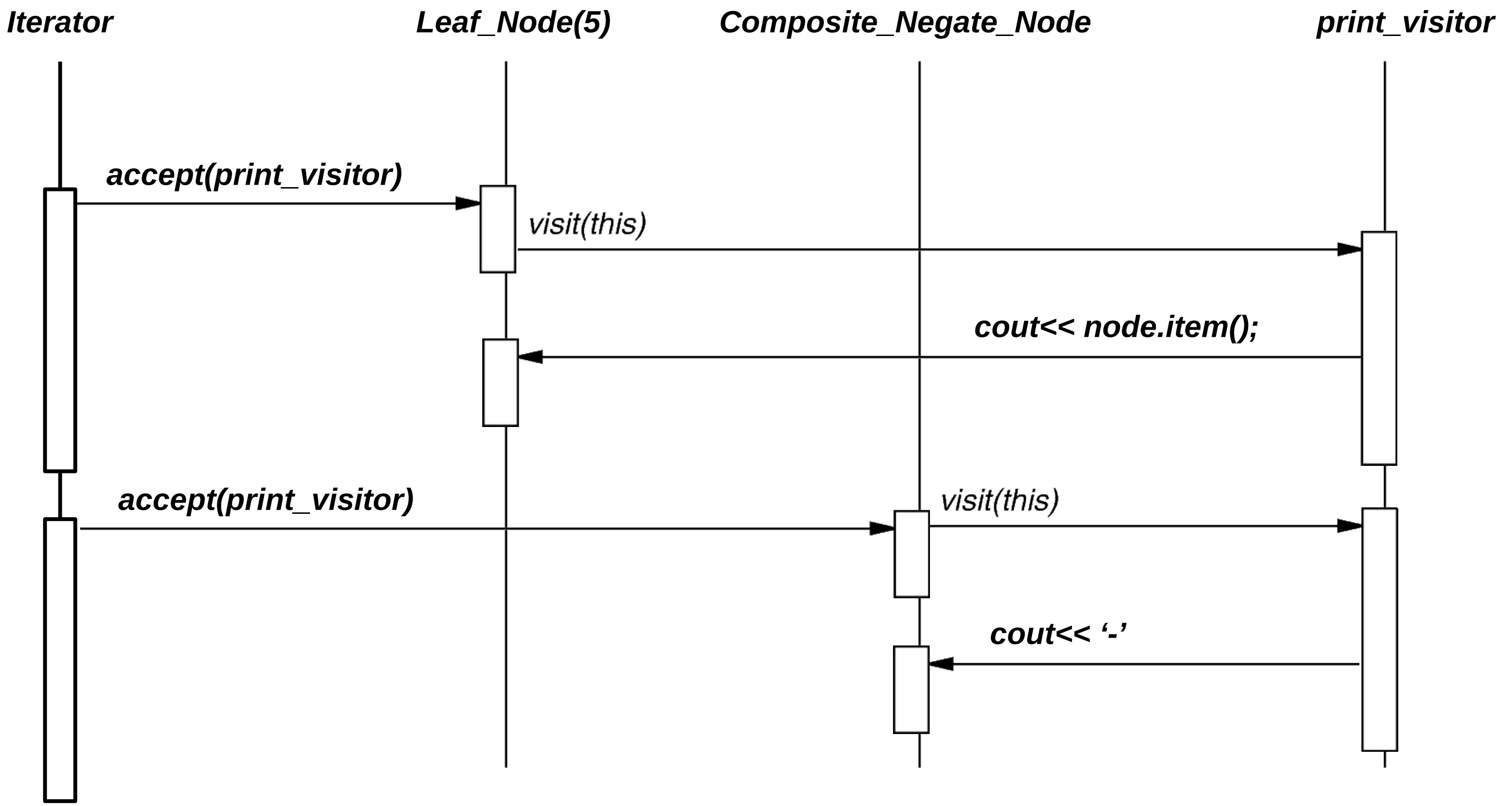
```
auto visitor = make_visitor ("print-visitor");

for (auto iter = expr_tree.begin("post-order");
     iter != expr_tree.end("post-order");
     ++iter)
  (*iter).accept(visitor);
```

← calls `visit(*this)`

Visitor GoF Object Behavioral

Visitor implementation in C++

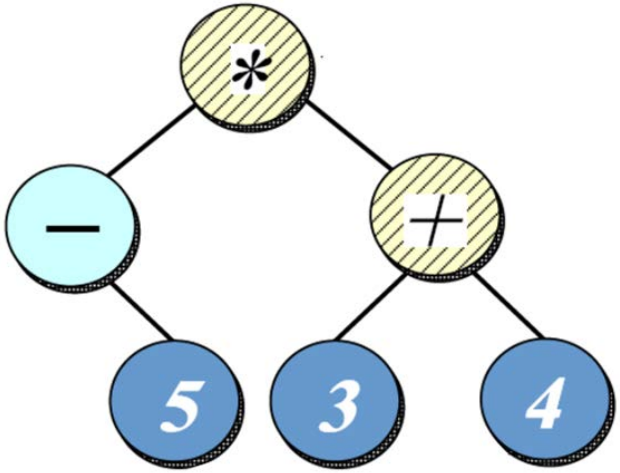
- The iterator controls the order in which **`accept()`** is called on each node in the composition
- **`accept()`** then "visits" the node to perform the desired print action



Visitor GoF Object Behavioral

Visitor implementation in C++

- The **Evaluation_Visitor** class evaluates nodes in an expression tree traversed using a post-order iterator
  - e.g., **5-34+***

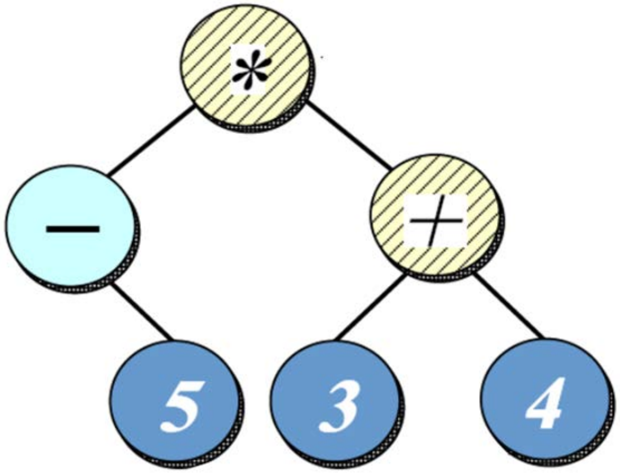


```
class Evaluation_Visitor
   : public ET_Visitor {
public:
    virtual void visit
         (const Leaf_Node &);
    virtual void visit
         (const Add_Node &);
    virtual void visit
         (const Divide_Node &);
    // etc.


};
```

Visitor GoF Object Behavioral

Visitor implementation in C++

- The **Evaluation_Visitor** class evaluates nodes in an expression tree traversed using a post-order iterator
  - e.g., **5-34+***



- It uses a stack to keep track of the post-order expression tree value that has been processed thus far during the iteration traversal

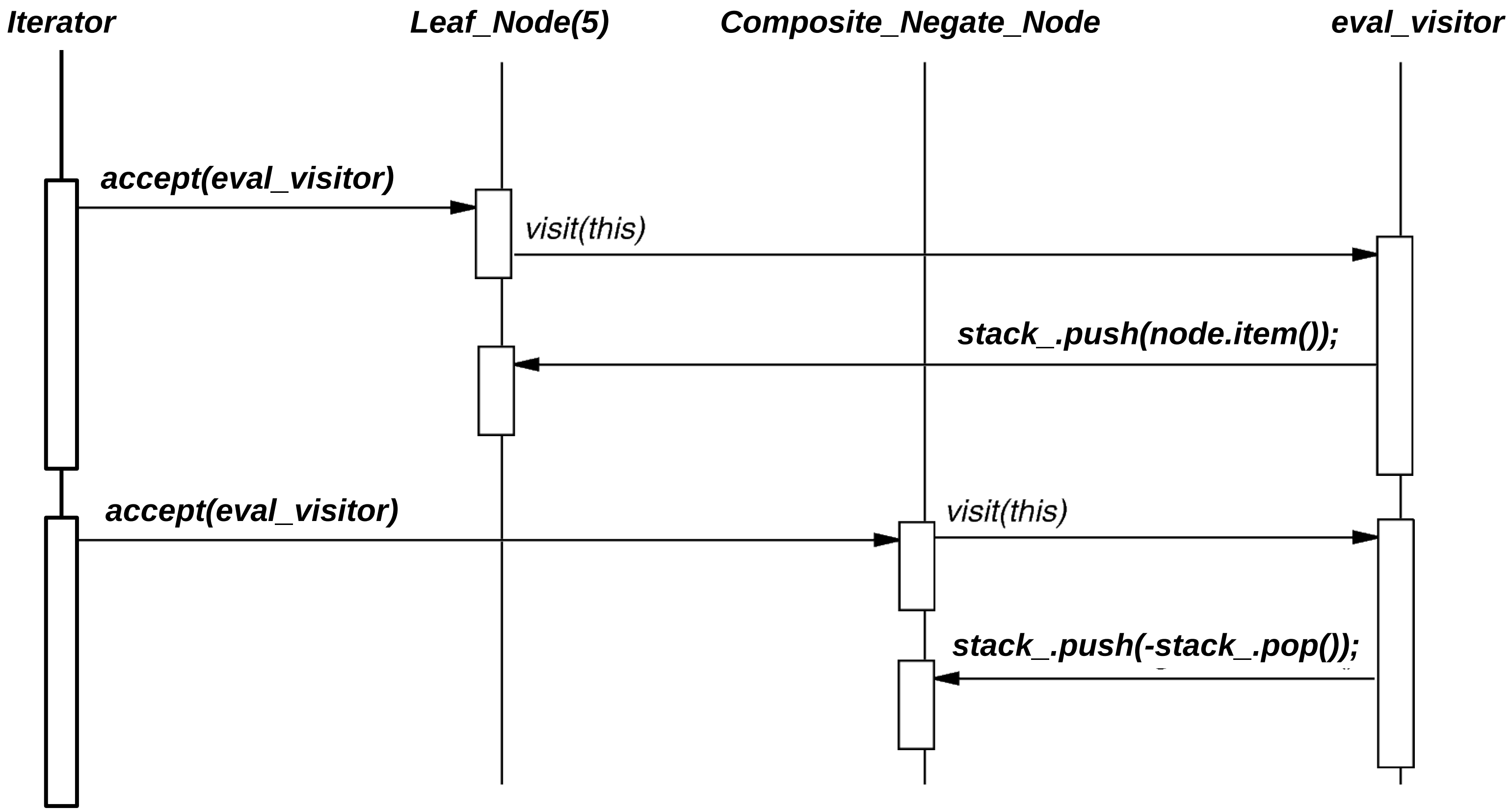
```
class Evaluation_Visitor
  : public ET_Visitor {
public:
    virtual void visit
         (const Leaf_Node &);
    virtual void visit
         (const Add_Node &);
    virtual void visit
         (const Divide_Node &);
    // etc.
private:
  std::stack<int> stack_;
};
```

1. S = [5]	**push(node.item())**
2. S = [-5]	**push(-pop())**
3. S = [-5, 3]	**push(node.item())**
4. S = [-5, 3, 4]	**push(node.item())**
5. S = [-5, 7]	**push(pop()+pop())**
6. S = [-35]	**push(pop()*pop())**

Visitor GoF Object Behavioral

Visitor implementation in C++

- The iterator controls the order in which **`accept()`** is called on each node in the composition
- **`accept()`** then “visits” the node to perform the desired evaluation action



Visitor GoF Object Behavioral

Consequences

+ *Flexibility*: Visitor algorithm(s) & object structure are independent

+ *Separation of concerns*: Localized functionality in the visitor subclass instance

– *Tight coupling*: Circular dependency between Visitor & Element interfaces

  – Visitor thus brittle to new ConcreteElement classes

Visitor GoF Object Behavioral

Consequences

+ *Flexibility*: Visitor algorithm(s) & object structure are independent

+ *Separation of concerns*: Localized functionality in the visitor subclass instance

– *Tight coupling*: Circular dependency between Visitor & Element interfaces

  – Visitor thus brittle to new ConcreteElement classes

Implementation

- Double dispatch (en.wikipedia.org/wiki/Double_dispatch)
- General interface to elements of object structure

Visitor GoF Object Behavioral

Consequences

+ *Flexibility*: Visitor algorithm(s) & object structure are independent

+ *Separation of concerns*: Localized functionality in the visitor subclass instance

– *Tight coupling*: Circular dependency between Visitor & Element interfaces

  – Visitor thus brittle to new ConcreteElement classes

Implementation

- Double dispatch (en.wikipedia.org/wiki/Double_dispatch)
- General interface to elements of object structure

Known Uses

- ProgramNodeEnumerator in Smalltalk-80 compiler
- IRIS Inventor scene rendering
- TAO IDL compiler to handle different backends