

Some discussions on Visitor

S.Ducasse, L. Fabresse, G. Polito, and P. Tesone



Goals

More on Visitor:

- Variations on navigation control
- Visitor detractors
- Visit methods granularity
- About double dispatch shortcutting
- ...



Controlling the traversal

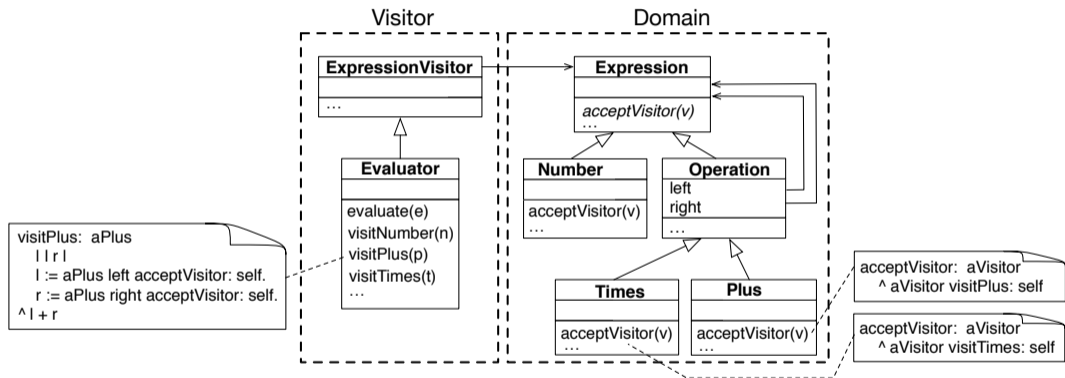
A visitor embeds a structure traversal that can be implemented:

- in the visitors
- in the domain elements themselves

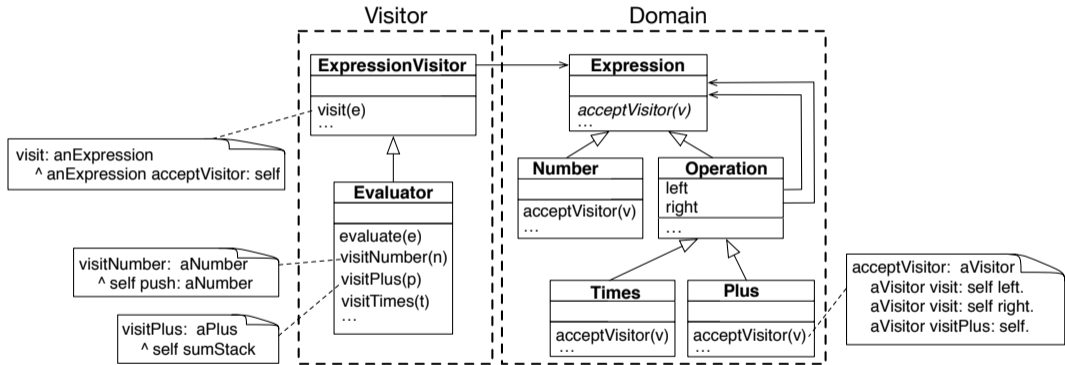
Usually the visitor controls the traversal but maybe the domain elements are more important



Visitor in control



Items in control



Visitor detractors

Visitor is not object-oriented because it externalizes behavior out of objects.

- Yes, operations applied to objects are defined outside themselves
- Are you ready to lose:
 - **clear separation** between operations related state and domain object state?
 - the possibility to **package multiple** behaviors **separately**?
 - the **incremental definition** of new operations?



Visitor vs. class extension

- Pharo supports class extension
 - i.e. defining methods on a class in another package than the class package

Should we use class extension instead of a Visitor?

- No, using a Visitor is better because:
 - Each Visitor **encapsulates a complex operation**
 - Each Visitor has its **own** state



Visit methods granularity

Compare these two Visitors:

| SimpleNodeVisitor |
|---------------------|
| visitNode(n) ... |

| ProgramNodeVisitor |
|---|
| visitNode(n) visitTemporaryVariable(n) visitLocalVariable(n) ... |

- SimpleNodeVisitor **only provides** visitNode which is **very high-level**
 - retrieving temporary variables would require testing and filtering nodes
- ProgramNodeVisitor **has richer API**
 - visitTemporaryVariable is **only invoked on temporaries**



Visit methods encode a context

- The granularity of visit methods has an impact
- Each **visit*** method provides a contextualized hook
- A too high-level API requires a lot of tests
- A too specialized API spreads information over multiple visit methods which is not good too
 - retrieving all variables involve a lot of hooks:
visitTemporaryVariable, visitLocalVariable, ...

| SimpleNodeVisitor |
|--------------------------|
| visitNode(n) ... |

| ProgramNodeVisitor |
|---|
| visitNode(n) visitTemporaryVariable(n) visitLocalVariable(n) ... |

About shortcutting the double dispatch

```
RBProgramNodeVisitor >> visitSequenceNode: aSequenceNode  
aSequenceNode statements do: [ :each | self visitVariable: each ]
```

Direct use of visitVariable:

- shortcuts the double dispatch
- does not let the domain decide
- prevents the use of a more specialized API: visitLocalVariable, visitTemporaryVariable, visitInstanceVariable

```
RBProgramNodeVisitor >> visitSequenceNode: aSequenceNode  
aSequenceNode statements do: [ :each |  
each acceptVisitor: self ]
```



Should we promote collections as domain nodes?

- When we iterate on a collection of nodes, the collection is not part of the composite domain
- Should we turn such a collection into a domain element?
- Not necessarily, it depends
 - can you change the domain?
 - think in terms of the benefit e.g., having the possibility to define `visitArrayOf...`



Building generic Visitors is difficult

There is no definitive solution. Usually, it is better to:

- have an abstract visitor
- redefine most of the logic per families of tasks



Visit methods and static types

Two alternatives to implement visit methods in statically typed languages:

- Using overloading
 - e.g., `visit(Number)`, `visit(Plus)`, `visit(Times)`
- Using different methods
 - e.g., `visitNumber(Number)`, `visitPlus(Plus)`, `visitTimes(Times)`

Avoid using overloading because:

- you will have to explicitly cast your objects everywhere
- you might have the wrong method executed (overload vs override)



Conclusion

- Visitor can be tricky to master
 - using `accept/visit` vocabulary helps readability
- Visitor is powerful for complex structure operations
 - it provides a **pluggable recursive treatment** of a composite structure



Produced as part of the course on <http://www.fun-mooc.fr>

Advanced Object-Oriented Design and Development with Pharo

A course by

S.Ducasse, L. Fabresse, G. Polito, and P. Tesone



Inria
LearningLab



Except where otherwise noted, this work is licensed under CC BY-NC-ND 3.0 France
<https://creativecommons.org/licenses/by-nc-nd/3.0/fr/>