Visitor Pattern – Behavioural
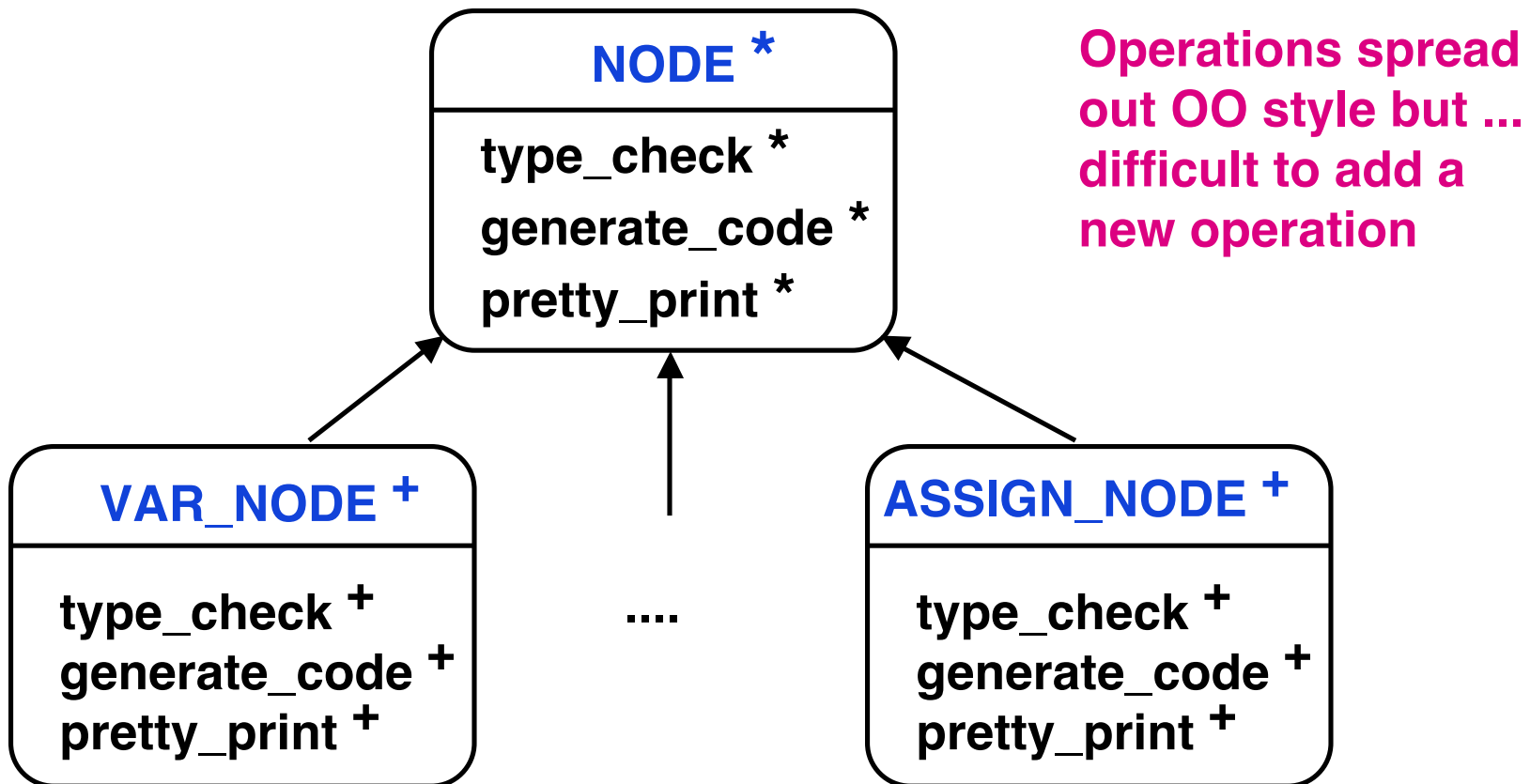
- Intent

    » **Represent an operation to be performed on all of the components of an object structure**

    » **Define new operations on a structure without changing the classes representing the components**
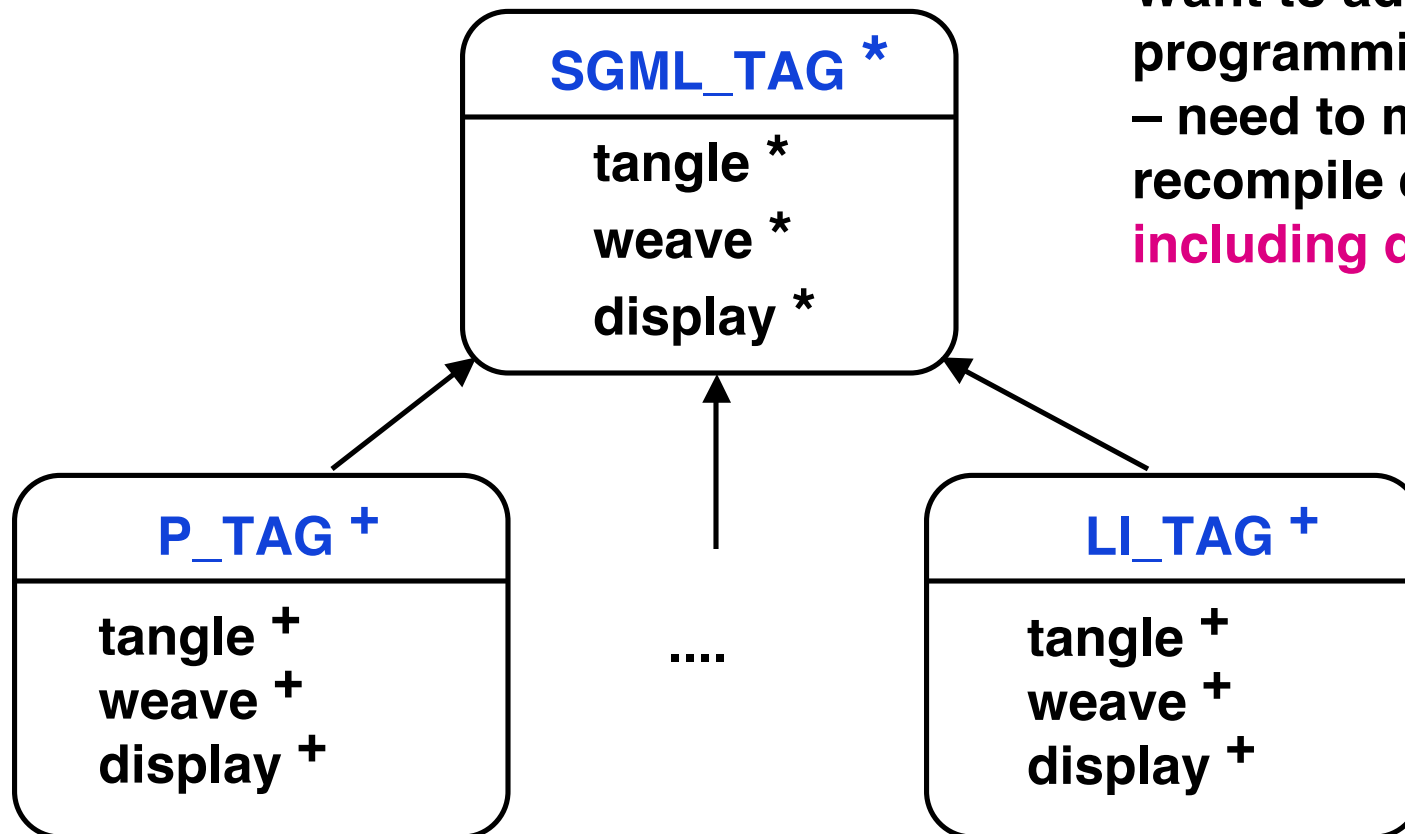
Visitor – Motivation

- Compiler using an abstract syntax tree

NODE *

type_check *

generate_code *

pretty_print *

Operations spread out OO style but ... difficult to add a new operation

VAR_NODE +

type_check +
generate_code +
pretty_print +

....

ASSIGN_NODE +

type_check +
generate_code +
pretty_print +

Visitor – Motivation – 2

- Consider programs to process SGML tags

Want to add literate programming processors – need to modify all classes recompile everything including deferred classes

SGML_TAG *

tangle *

weave *

display *

P_TAG +

tangle +
weave +
display +

....

LI_TAG +

tangle +
weave +
display +

Visitor Architecture – SGML Example

- Tag classes are independent of every operation

- Nodes accept visitors and direct them to the appropriate operation – through polymorphism

* VISITOR

* SGML_TAG

+ PS_GENERATE

+ LI_TAG
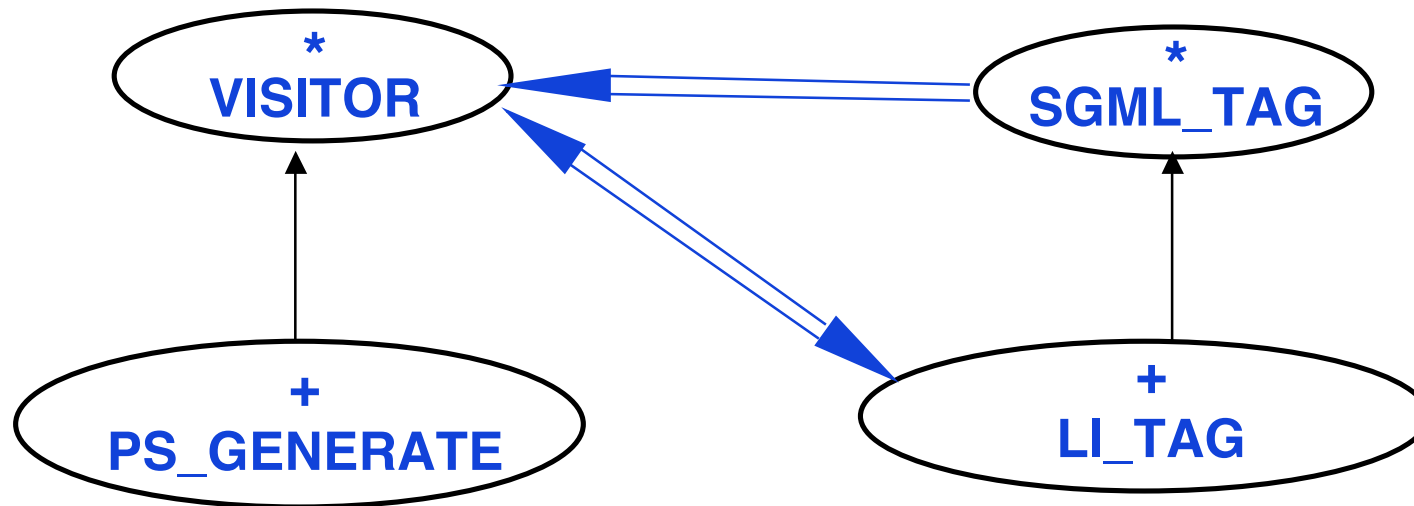
One visitor subclass for each operation

Same structure for each tag

Visitor – Applicability

- Object structure contains many classes of objects with differing interfaces and want to perform operations that depend on their concrete classes

- Many distinct and unrelated operations need to be performed

    » **Do not want to or are unable to clutter the concrete classes with these operations**

    » **Keep the related sub-operations (specific to each concrete class) together**

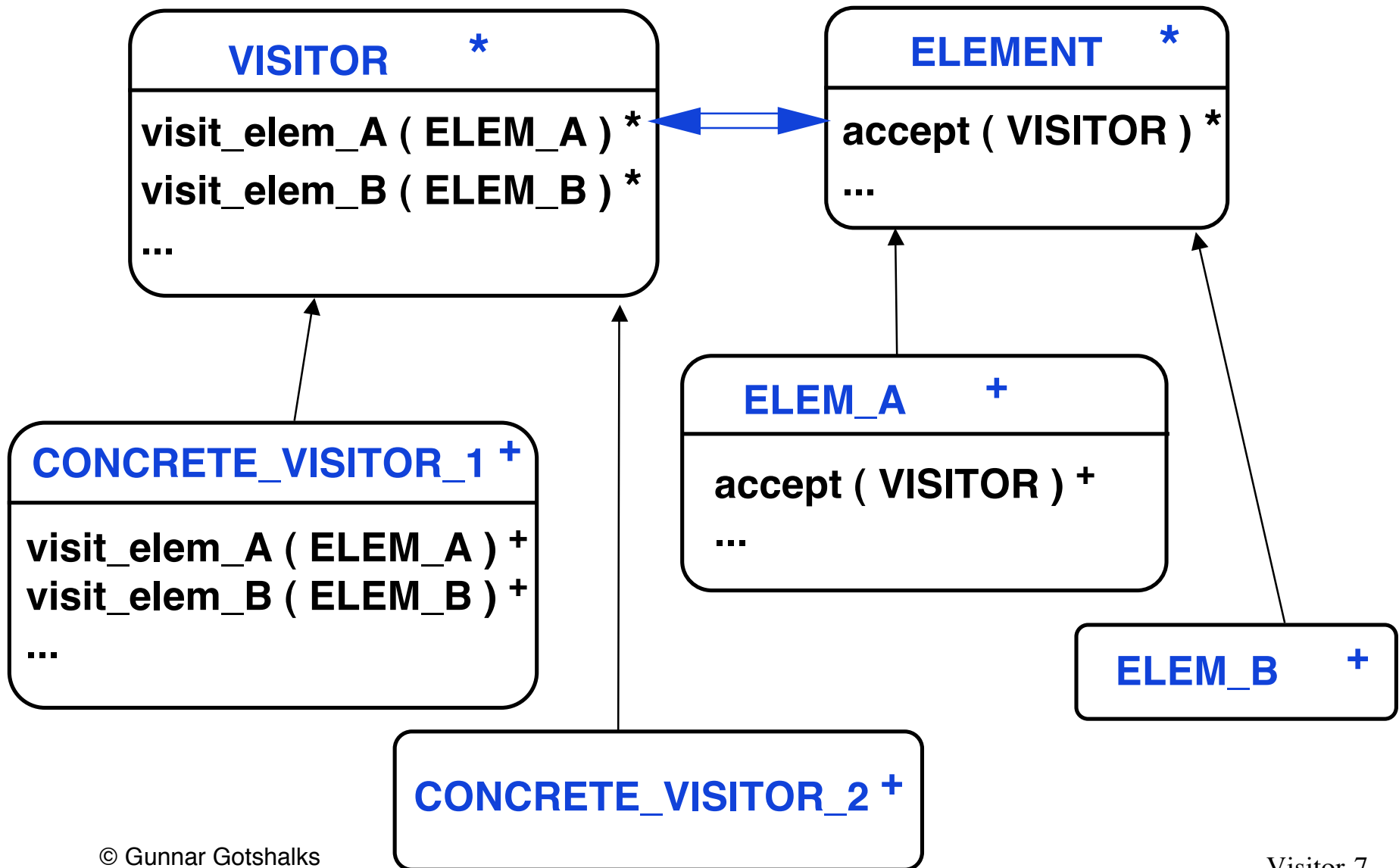   » **Put operations into only those applications that need them**

Visitor – Applicability – 2

• The classes defining the object structure rarely change, but you often want to define new operations over the structure

Changing object structure means

Redefining interface to all visitors, which is costly

Visitor – Abstract Architecture

VISITOR *

visit_elem_A (ELEM_A) *
visit_elem_B (ELEM_B) *
...

⟷

ELEMENT *

accept (VISITOR) *
...

CONCRETE_VISITOR_1 +

visit_elem_A (ELEM_A) +
visit_elem_B (ELEM_B) +
...

ELEM_A +

accept (VISITOR) +
...

ELEM_B +

CONCRETE_VISITOR_2 +

Visitor – Participants

- Element

    Declares accept method for Visitors

- Concrete element

    Implements accept method for Visitors

- Visitor

    Declares visit operation for each concrete element class

Visitor – Participants – 2

- Concrete visitor

  - » **Implements every visit operation declared by the Visitor**

    Each visit operation implements a fragment of the algorithm defined for the concrete visitor

  - » **Provides the context for the algorithm composed of all of the visit fragments**

    State accumulates with each visit

  - » **Implements the high level organization**

    - > **Iteration over the components**
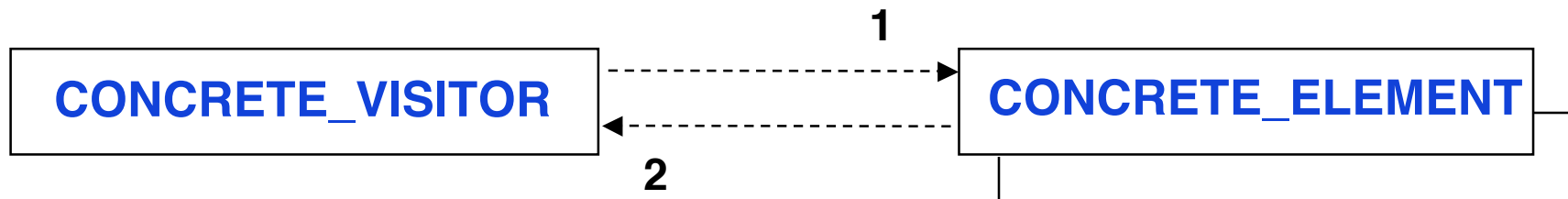
    - > **Processing each in turn**

Visitor – Scenario

- Concrete visitor loops over the elements

- For each element concrete visitor

Selects which method in the visitor to execute

Scenario: **Do one visit**
1 **accept (concrete_visitor)** 2 **visit_routine (concrete_element)**

1

CONCRETE_VISITOR ⇢ **CONCRETE_ELEMENT**

2

Visitor – TAG Implementation

deferred class SGML_TAG feature

    accept (visitor : VISITOR) deferred end

 ... -- other features ...

end

  class LI_TAG inherit SGML_TAG feature

    accept (visitor : VISITOR) do
      visitor.visit_li_tag (Current)
   end

 ... -- other features ...

 end

Visitor – VISITOR Implementation

deferred class VISITOR feature

   -- Have one "visit" routine for each tag (component)

   visit_LI_TAG (tag : LI_TAG) deferred end
   visit_P_TAG (tag : P_TAG) deferred end
   visit_UL_TAG (tag : UL_TAG) deferred end
   ...
end

Visitor – Concrete Visitor Implementation

```
class CONCRETE_VISITOR inherit  VISITOR feature

    get_elements     -- Attaches elements to the iterator
    while not elements.allDone do
        elements.item.accept ( Current )
        elements.next
    end

    visit_li_tag (tag : LI_tag) do semantic action ... end
    visit_ul_tag (tag : UL_tag) do semantic action ... end
    visit_p_tag (tag : P_tag) do semantic action ... end

    ... -- and all the rest of the tags

end
```
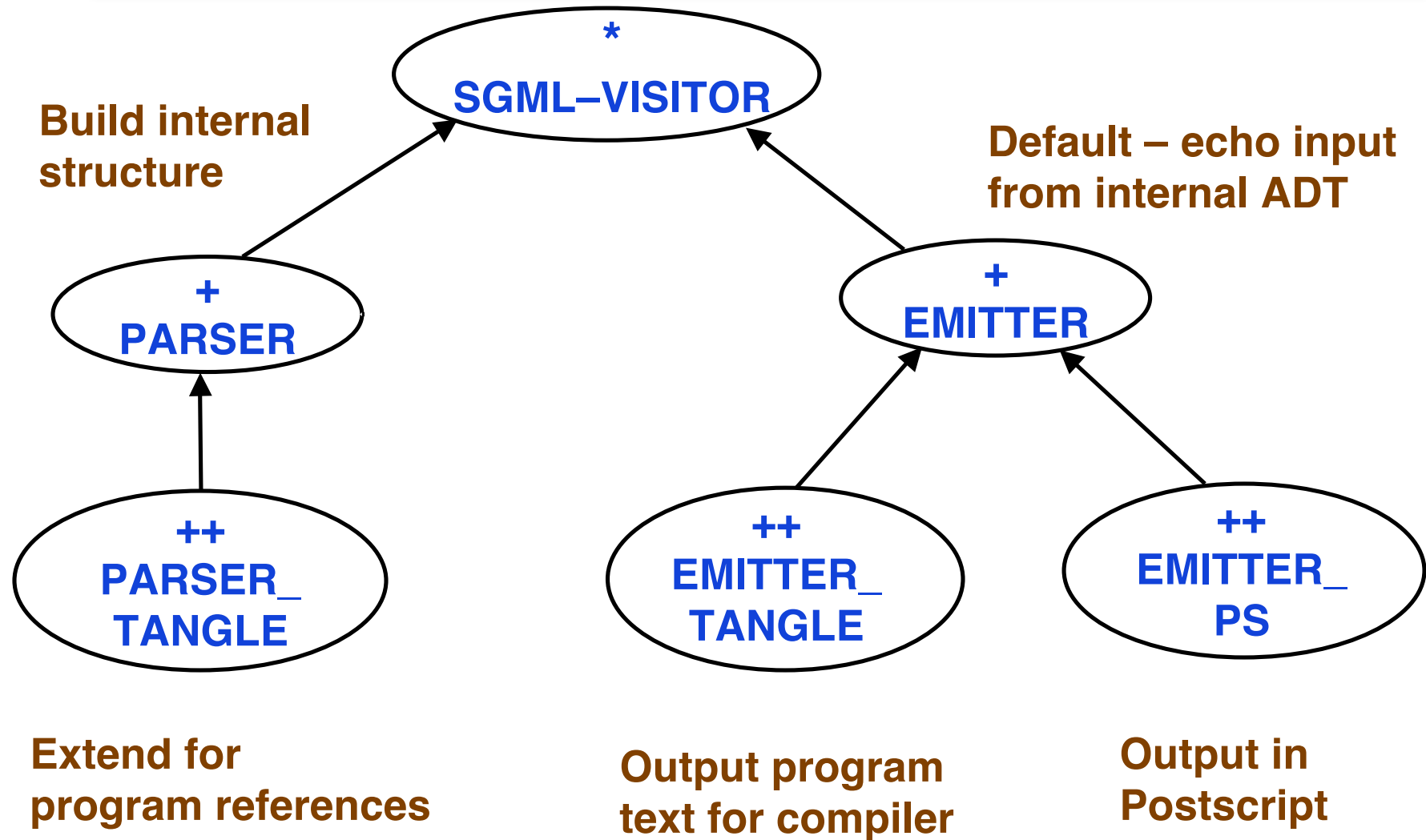
Visitor – Consequences

- Adding new operations is easy

  » **New operation implements visitor interface for the components**

  » **All the fragments of the visitor algorithm are in one file – related behaviours are together**

    Easier to make sure that components are working in unison

  » **Unrelated operations and fragments are in other visitor classes**

  » **Contrast with having to change each of the component classes to have the operation fragment**

    Each class has a fragment of each of the operations

Visitor – Consequences – 2

- Adding new concrete elements is difficult

  » **Need to modify visitor class**

  » **Need to modify each concrete visitor**

    > **Can sometimes simplify as many elements have common behaviour (default behaviour) that can be specified at concrete visitor level 1.**

    > **Create subclasses of level 1 for more specific behaviour for the new elements**

      – **Only program the new elements**

  » **For many structures components do not change rapidly so this is not a problem**

Example Multi-Level Visitor

*
SGML−VISITOR

**Build internal
structure**

**Default – echo input
from internal ADT**

**+
PARSER**

**+
EMITTER**

**++
PARSER_
TANGLE**

**++
EMITTER_
TANGLE**

**++
EMITTER_
PS**

**Extend for
program references**

**Output program
text for compiler**

**Output in
Postscript**

Visitor – Consequences – 3

- Works across class hierarchies

  » **Contrast with Iterator Pattern**

  » **Contrast with multi-panel & do-undo applications**

Visitor – Related Patterns

- Visitor pattern is used to apply an operation over a Composite

- Visitor pattern is used to do the interpretation for an Interpreter pattern

- Visitor pattern is used to process the items obtained by using the Iteration pattern to iterate over a collection.