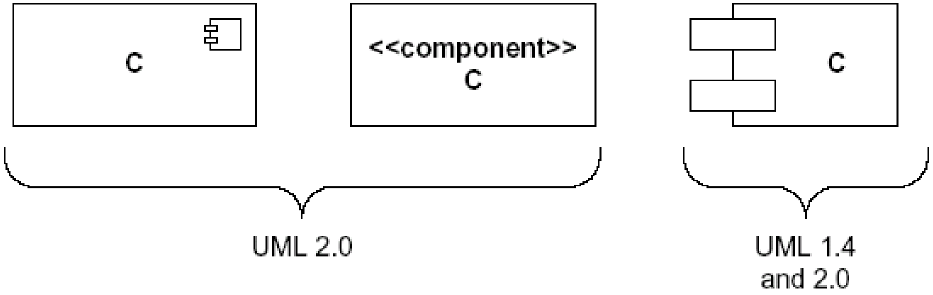
Component Diagrams

UNIVERSITÄT SALZBURG

Components

Classes can be grouped in components. In UML, a component can be represented as follows:



Components correspond to modules in module-oriented languages.

C++: Reproduction of modules through .h, .c files

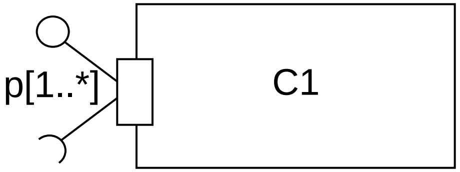
Smalltalk: Groups of classes, no modules

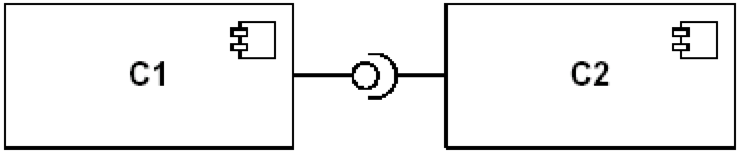
Oberon and Java: Modularity supported directly by the language

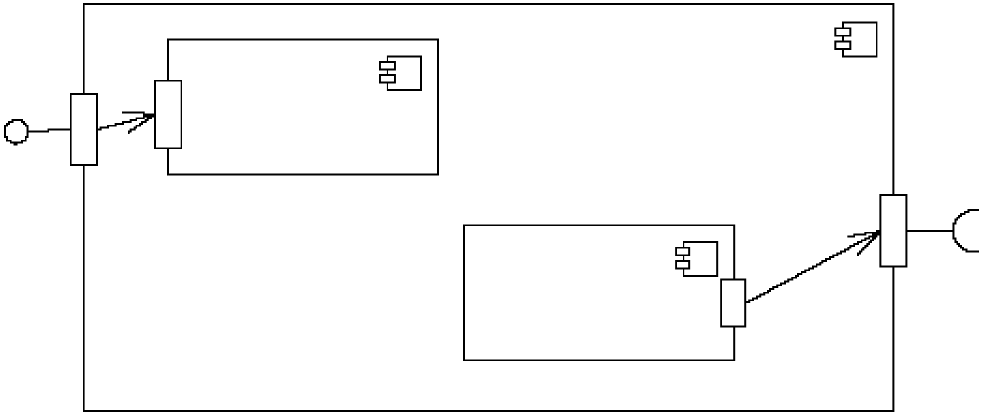
Ports, Interfaces and Connectors

- Ports: interaction points
- Interfaces:
  - Provided
  - Required

- Connectors:
  - Assembly
  - Delegation





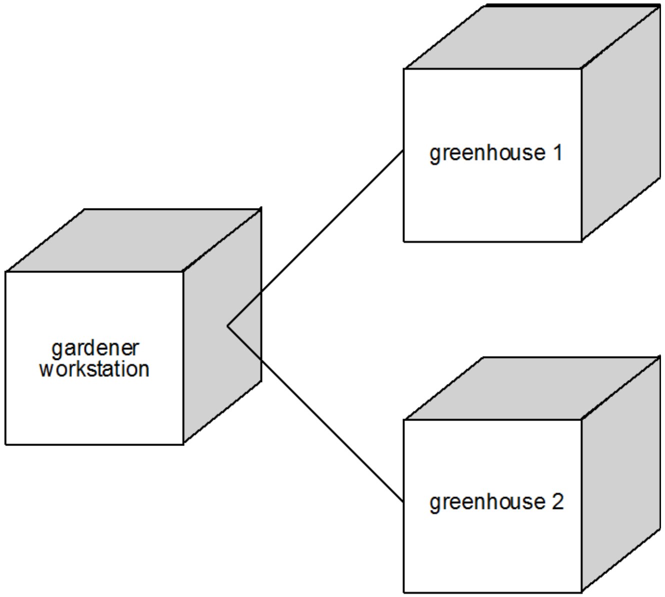


Deployment Diagrams

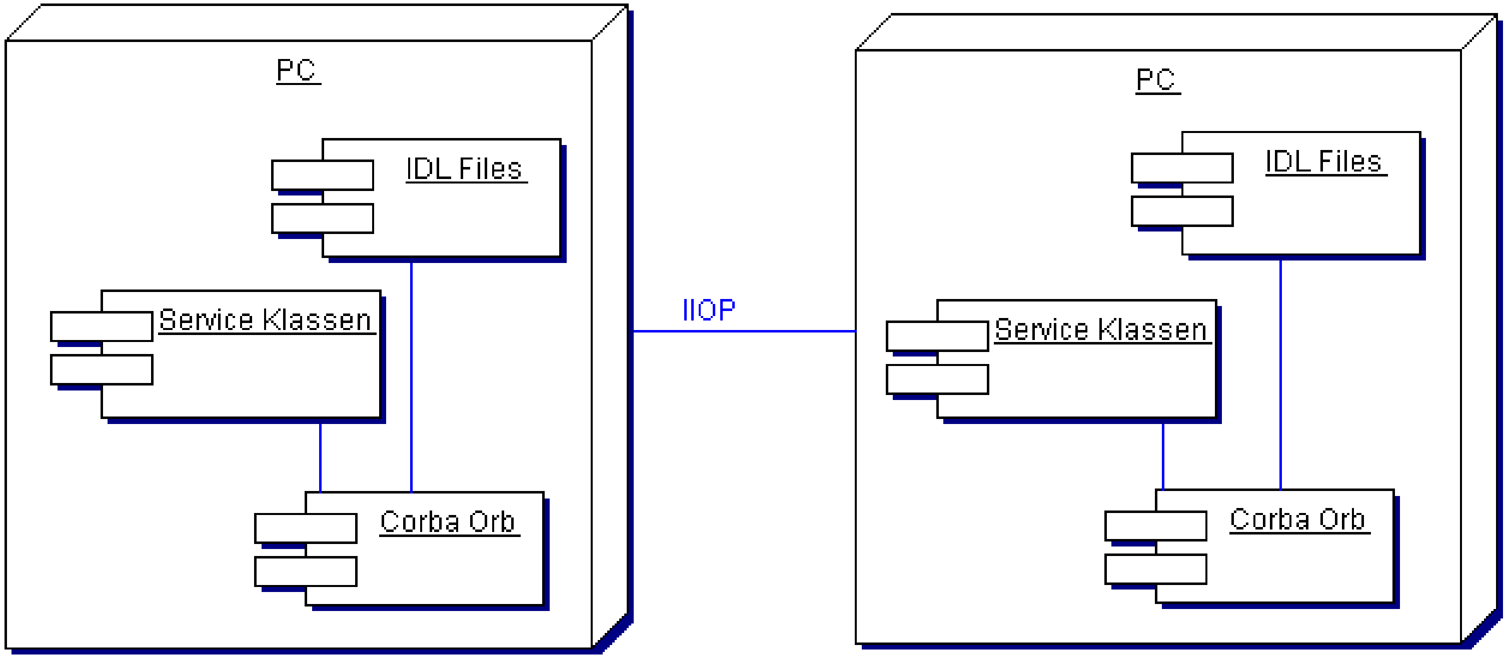
UNIVERSITÄT SALZBURG

Notation

This representation is developed from Booch' s process diagram. It expresses the assignment of main programs and/or active objects to processors **for distributed systems running on multiple processors**.



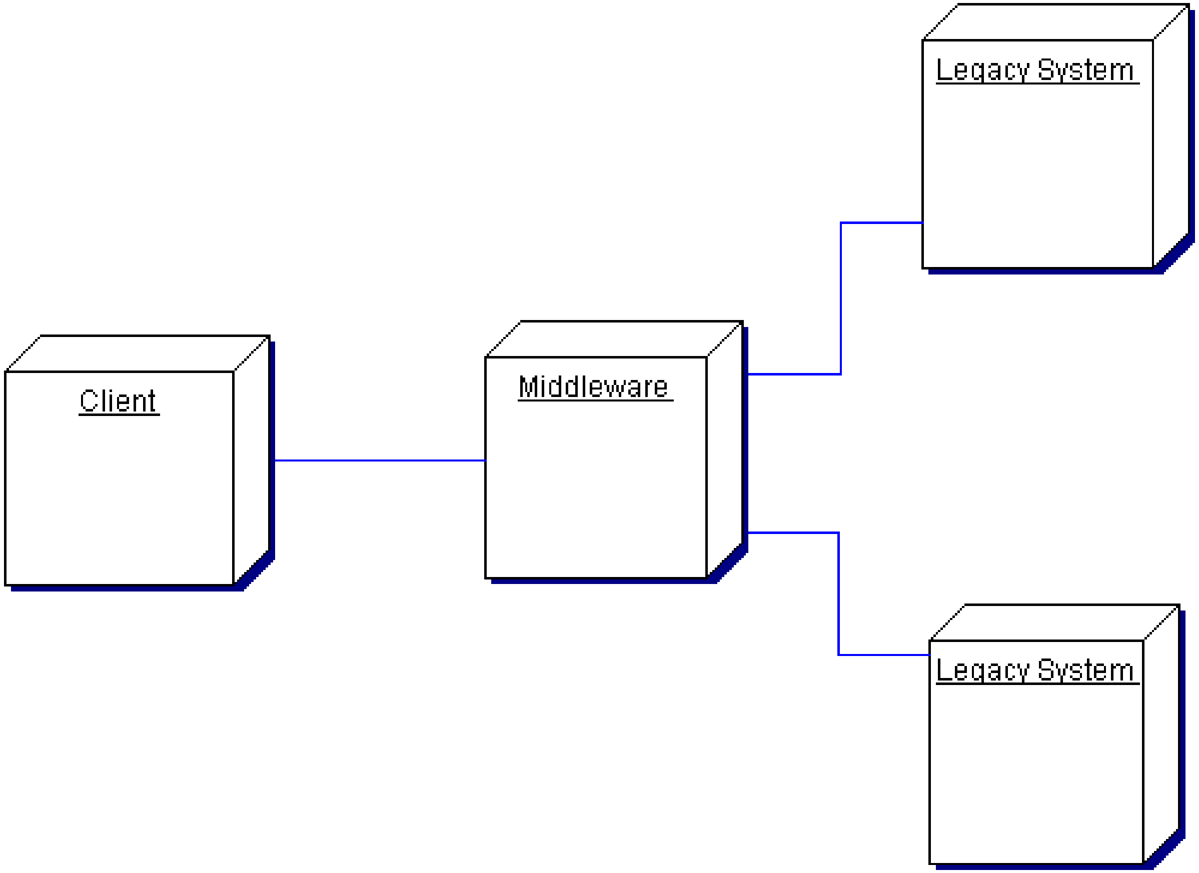
Example: CORBA



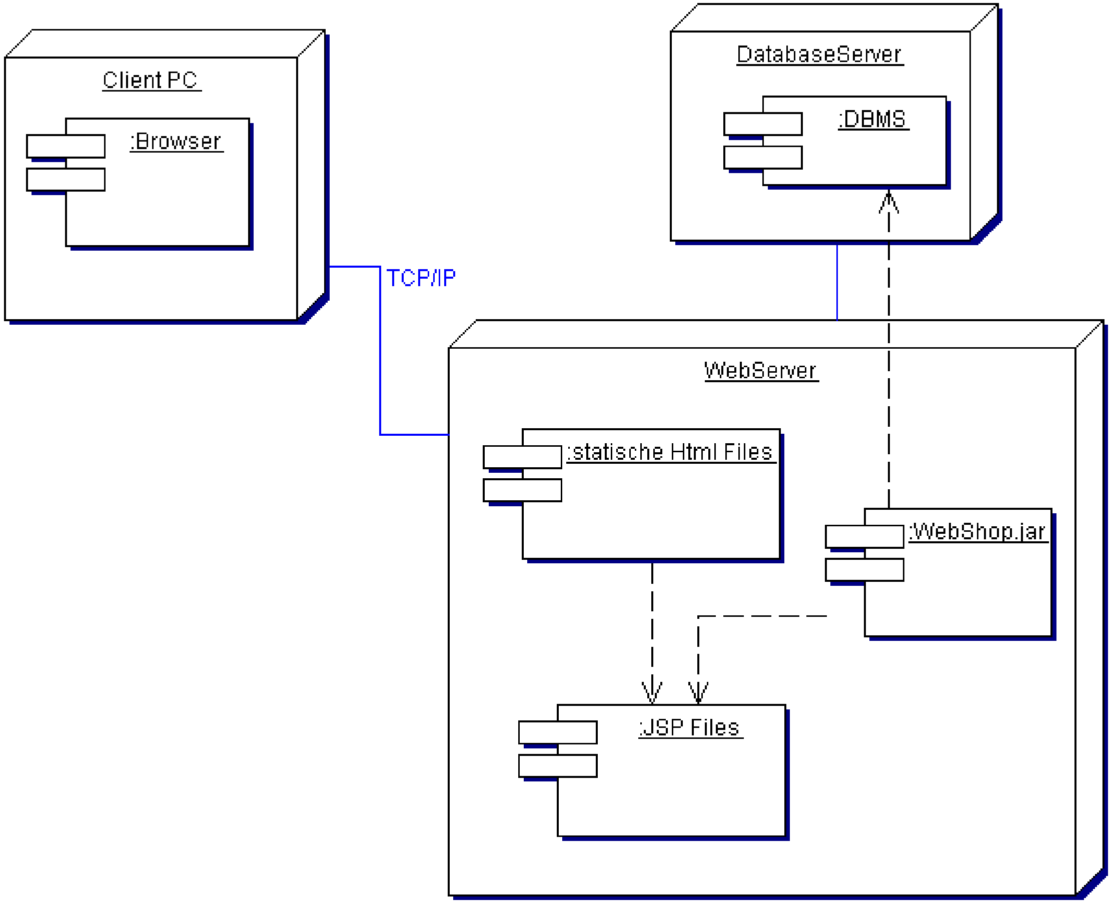
Hands-On Exercise: Web Shop

- A Webshop is typically a distributed application, where multiple layers are involved.
- How could the topology of the system look?
- Which components are on which computational nodes?

Three-tier Architecture



Web Shop: Topology



Construction of Flexible Software

UNIVERSITÄT SALZBURG

Contents

- Configuration parameters
- Concepts and contruction principles for flexible, object-oriented product families
- Design Patterns

UNIVERSITÄT SALZBURG

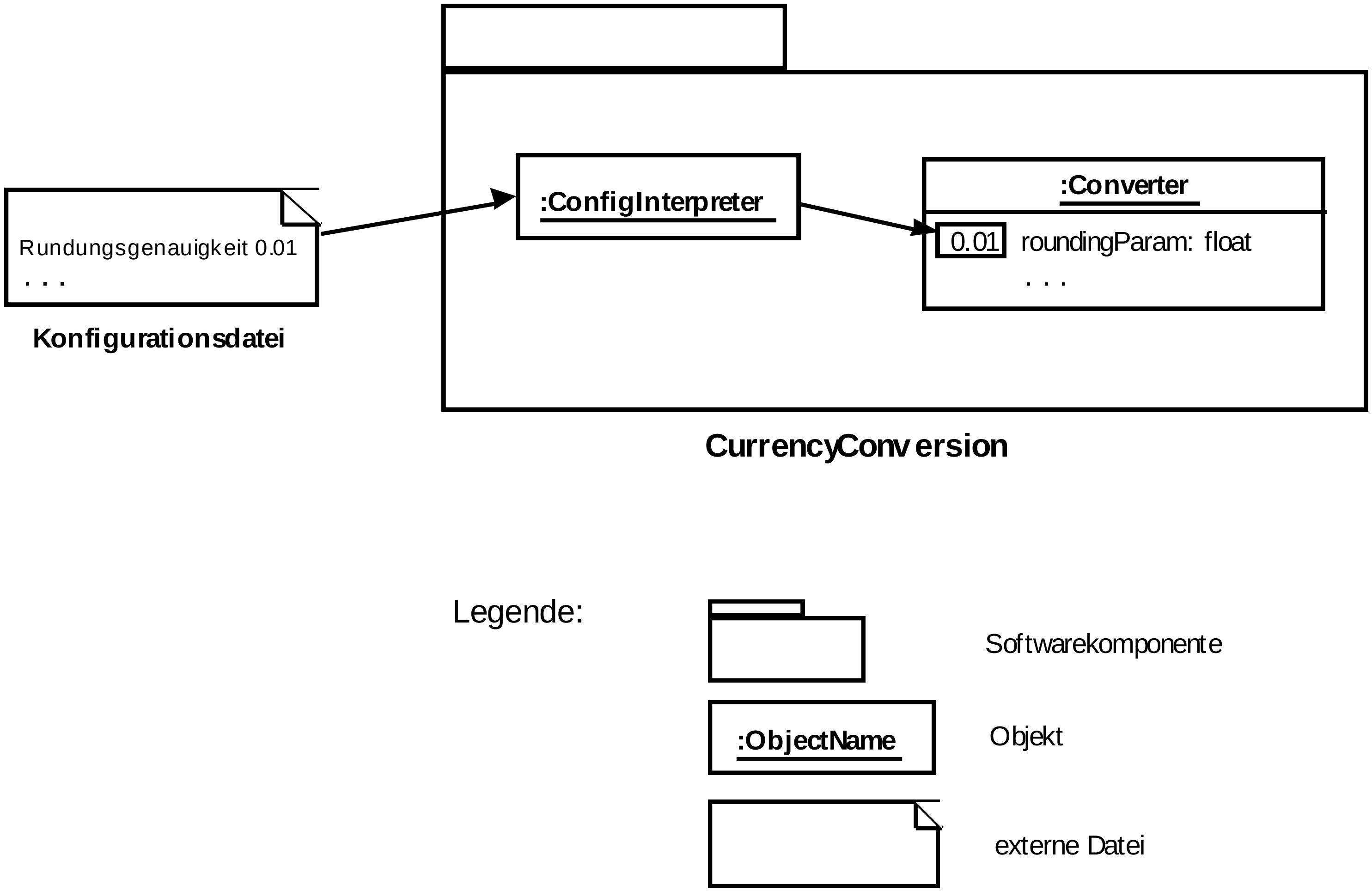
Configuration

Definition

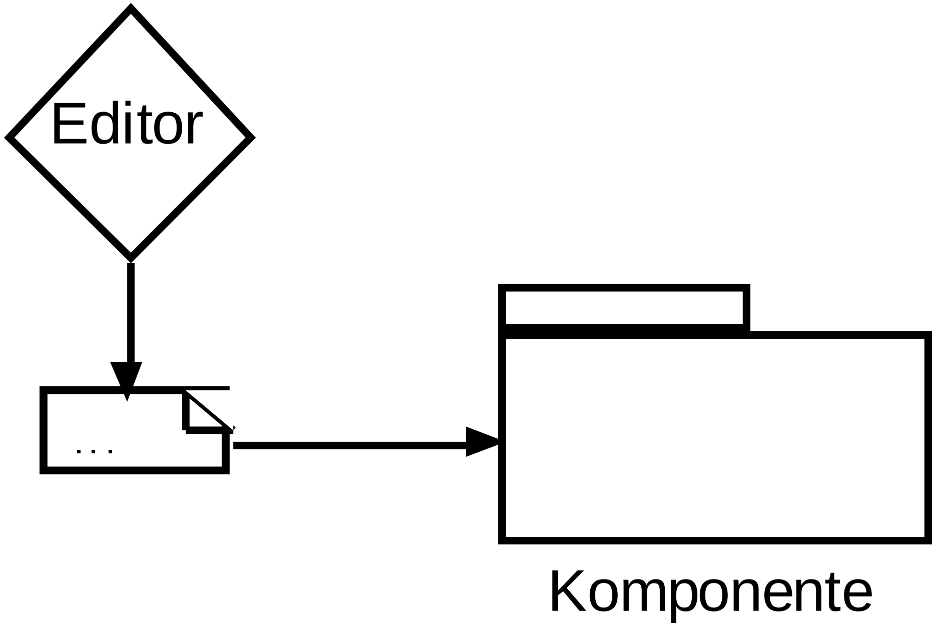
- Configuration parameters are placed in configuration files.

- Configuration parameters correspond to persistent, global (= static) variables.

Example


Rundungsgenauigkeit 0.01
. . .
Konfigurationsdatei
:ConfigInterpreter
:Converter
0.01
roundingParam: float
. . .
CurrencyConversion
Legende:
Softwarekomponente
:ObjectName
Objekt
externe Datei

Generating the Configuration File



Example:

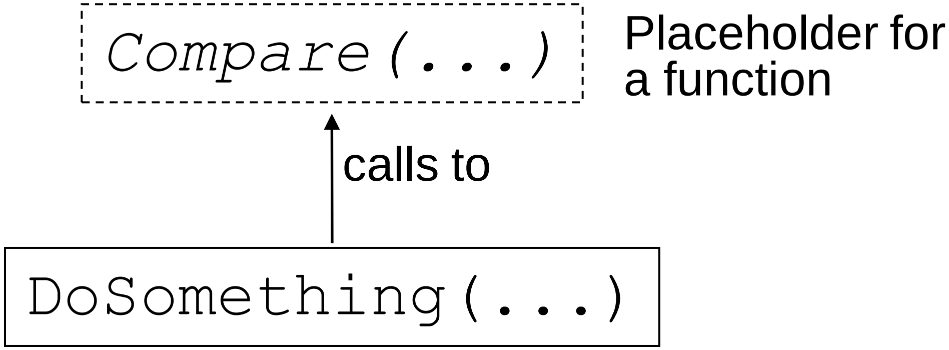
GUI Configuration file = Resource file

Visual, interactive construction with help from resource editors

Concepts and Construction Principles for Flexible Object-Oriented Product Families

UNIVERSITÄT SALZBURG

The Callback Style of Programming (I)



DoSomething calls a function which it has received as an argument. This shows the meaning of the callback style of programming:

One can conceptually distinguish whether a function or a procedure is **called directly (call)** or whether a function or a procedure passed as a parameter is **called indirectly (by means of callback).**

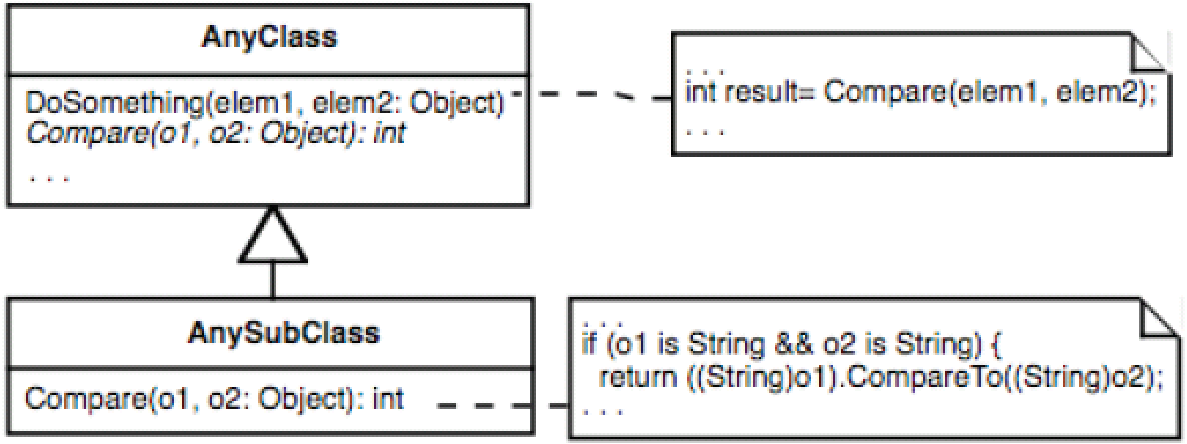
The Callback Style of Programming(II)

```
void DoSomething(int (*Compare)(void*, void*),
                                void* elem1, void* elem2 )
```

```
int StringCompare(void* string1, void* string2) {
            return strcmp(    // C-Bibliotheksfunktion strcmp
                                    (char*)string1,
                                    (char*)string2
                        );
    }  // StringCompare
```

```
DoSomething(StringCompare, “first“, “second“);
```

The Callback Style of Programming(III)



AnyClass

DoSomething(elem1, elem2: Object)
Compare(o1, o2: Object): int
. . .

```
. . .
int result= Compare(elem1, elem2);
. . .
```

AnySubClass

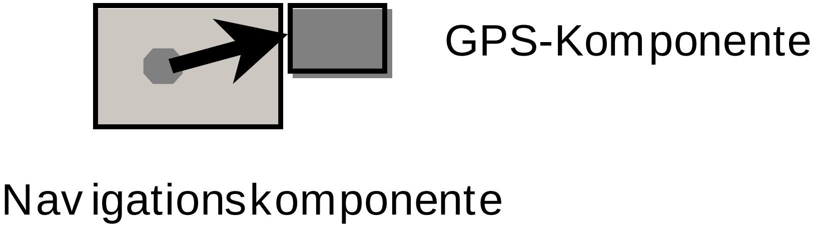
Compare(o1, o2: Object): int

```
. . .
if (o1 is String && o2 is String) {
  return ((String)o1).CompareTo((String)o2);
. . .
```

Definition

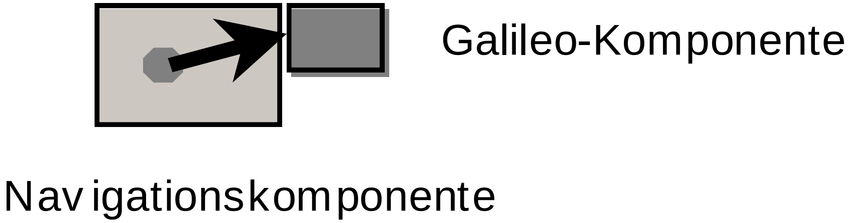
Product Family: A piece of software from which different applications can be formed by the callback style of programming, i.e. its behavior is changeable and/or expandable.

Abstract Coupling



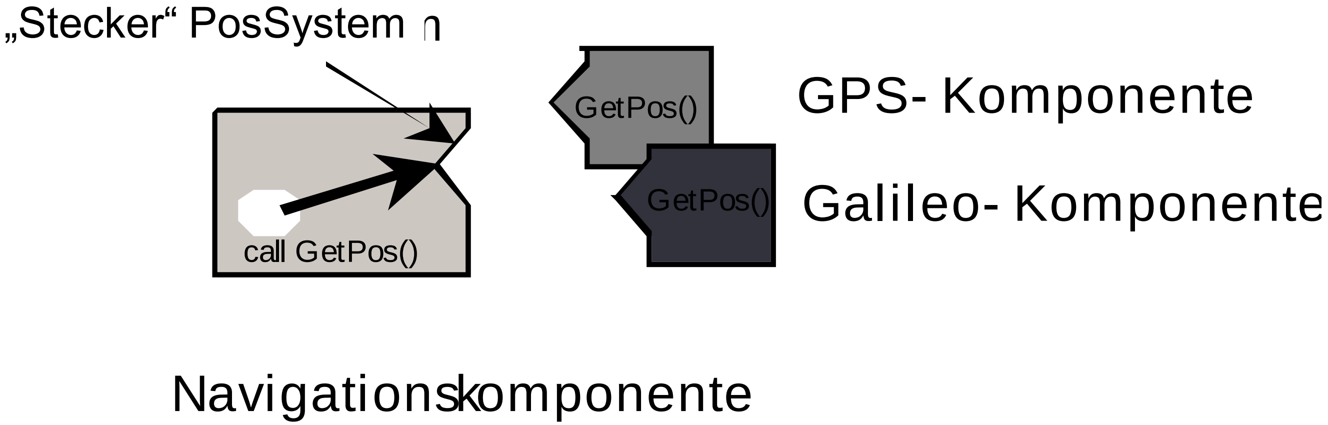
GPS-Komponente

Navigationskomponente



Galileo-Komponente

Navigationskomponente



„Stecker“ PosSystem

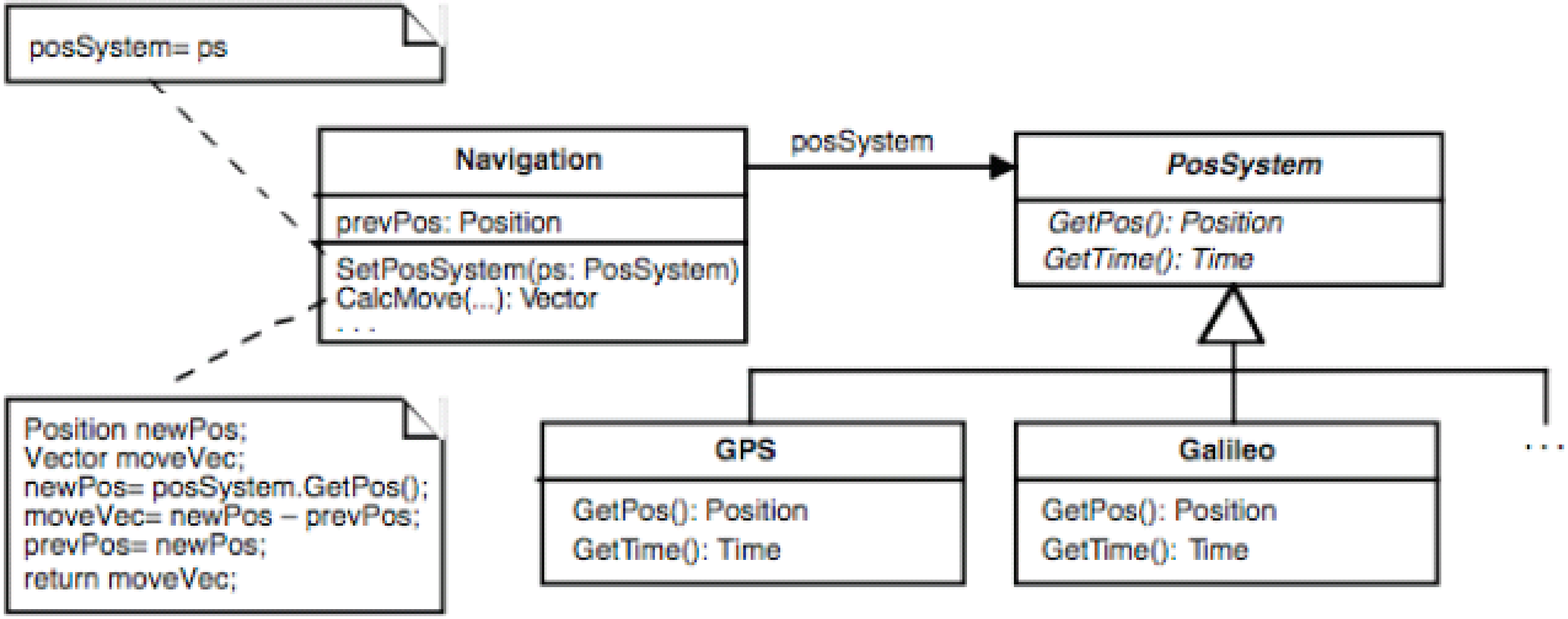
GPS- Komponente

Galileo- Komponente

Navigationskomponente

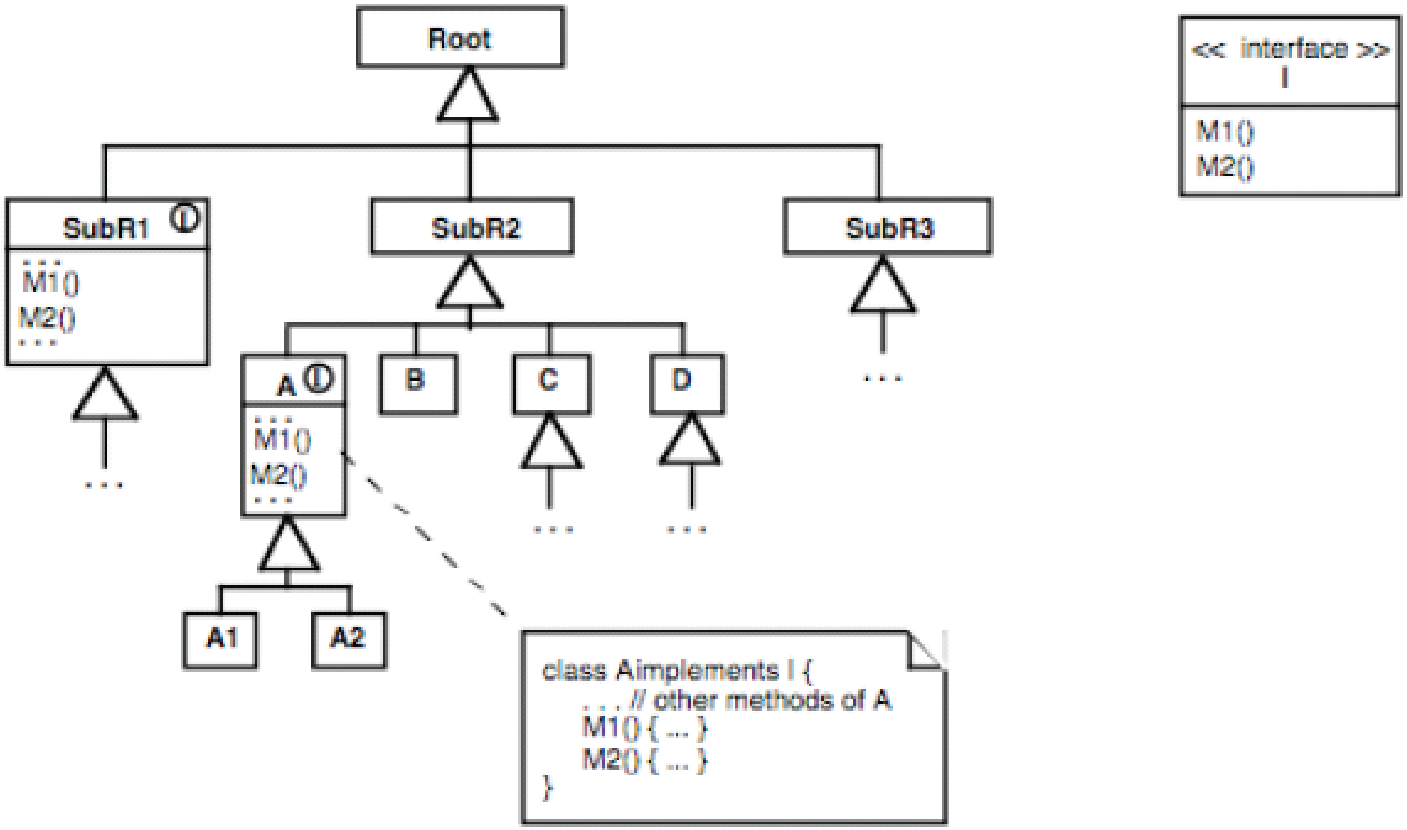
Abstract Coupling by Abstract Classes

Navigation system example:



```
Position newPos;
Vector moveVec;
newPos= posSystem.GetPos();
moveVec= newPos – prevPos;
prevPos= newPos;
return moveVec;
```

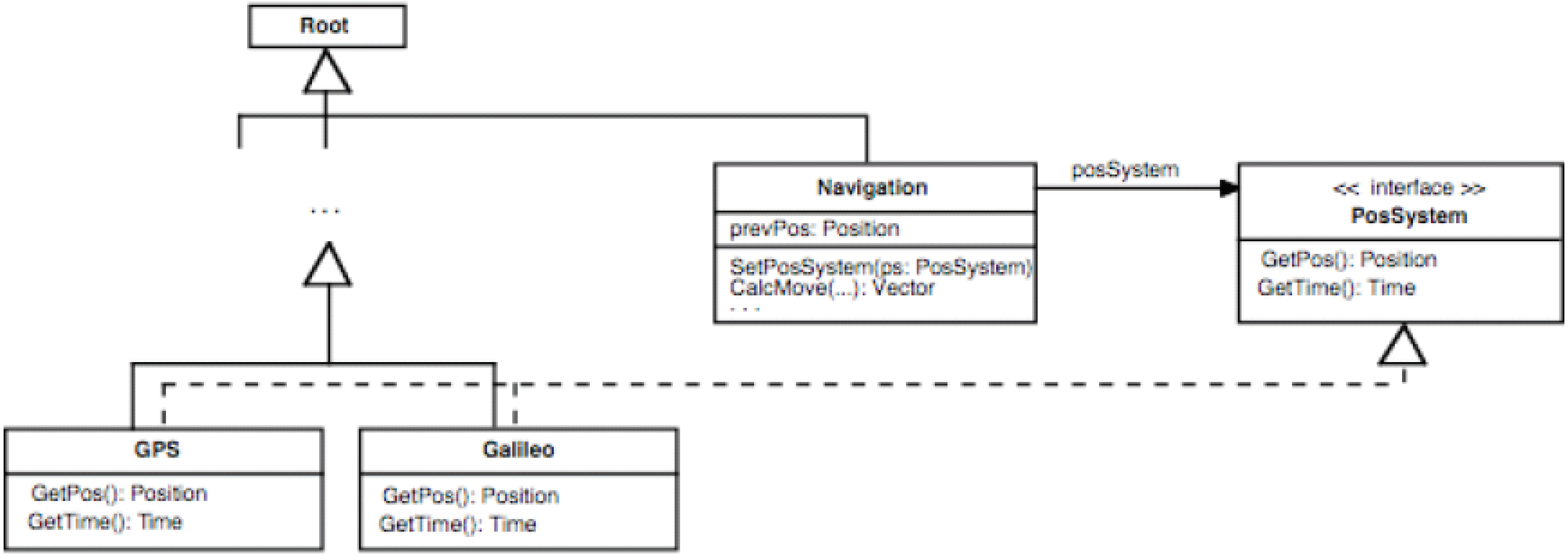
Alternative: Interfaces



```
class Aimplements I {
    . . . // other methods of A
    M1() { ... }
    M2() { ... }
}
```

Abstract Coupling by Interfaces

Navigation system example:



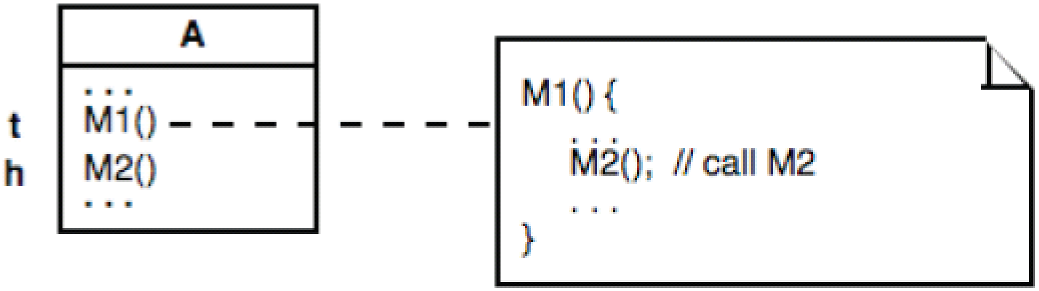
Template and Hook Methods

Definition

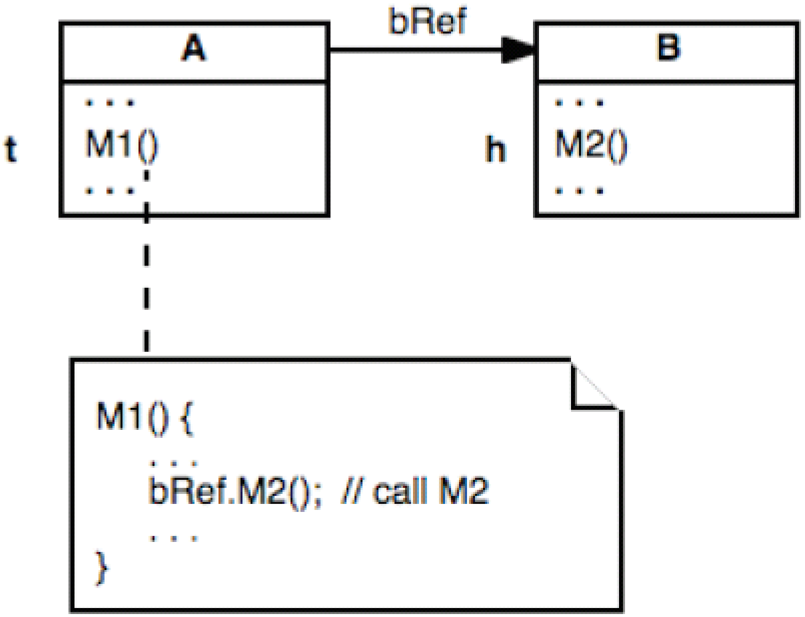
If a method is called in another method’s implementation, then we call the **calling method the Template method** and the **called method the Hook method**.

The template method addressed here has nothing to do with the C++ language construct `template`.

Both Methods in the Same Class

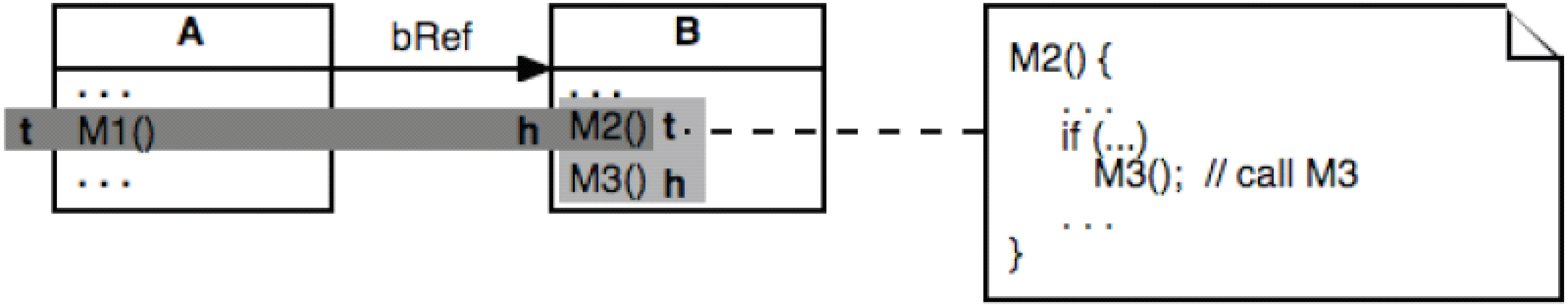


Template and Hook Methods in Different Classes

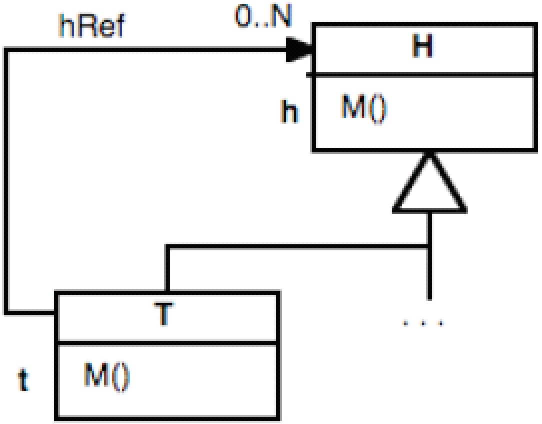


```
M1() {
    . . .
    bRef.M2();  // call M2
    . . .
}
```

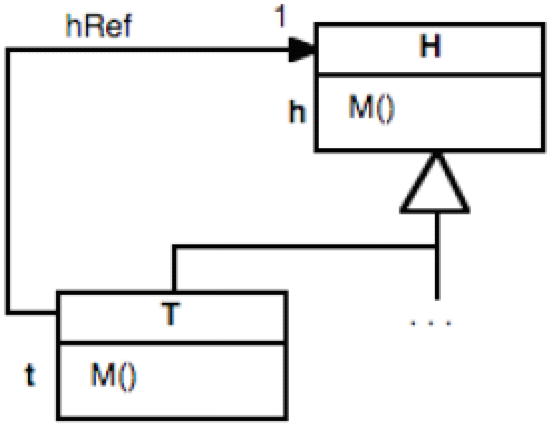
The same method can be both Template and Hook depending on the context



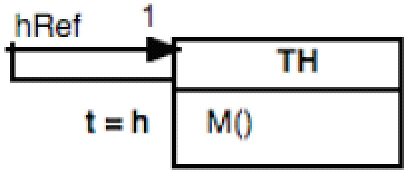
Combinations With Recursiveness



Composite



Decorator



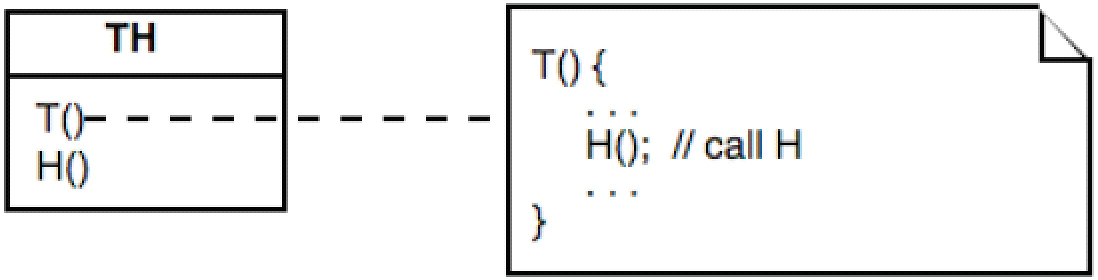
Chain-Of-Responsibility

UNIVERSITÄT SALZBURG

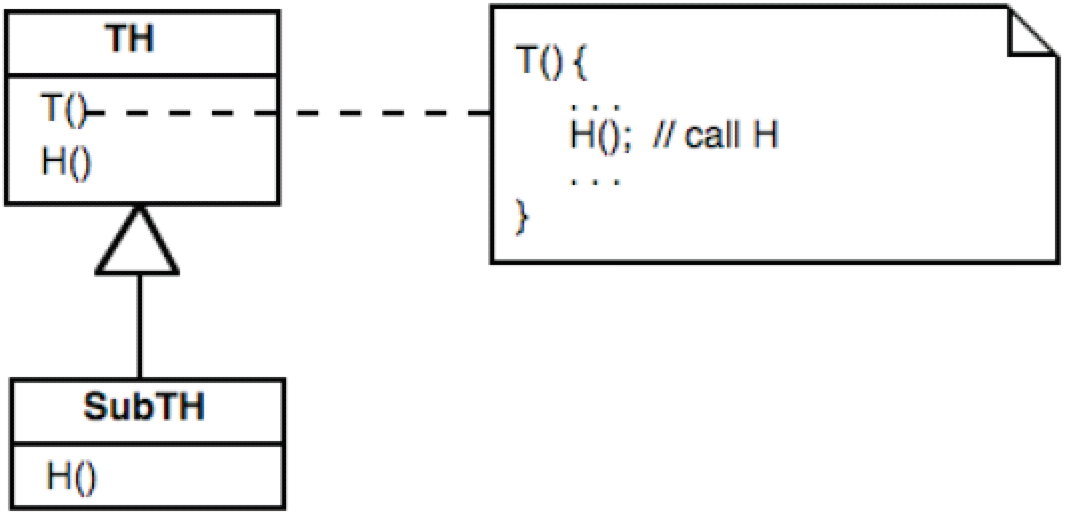
Hook Method Construction Principle

UNIVERSITÄT SALZBURG

Hook Method: Adaptation of T() by overwriting of H()

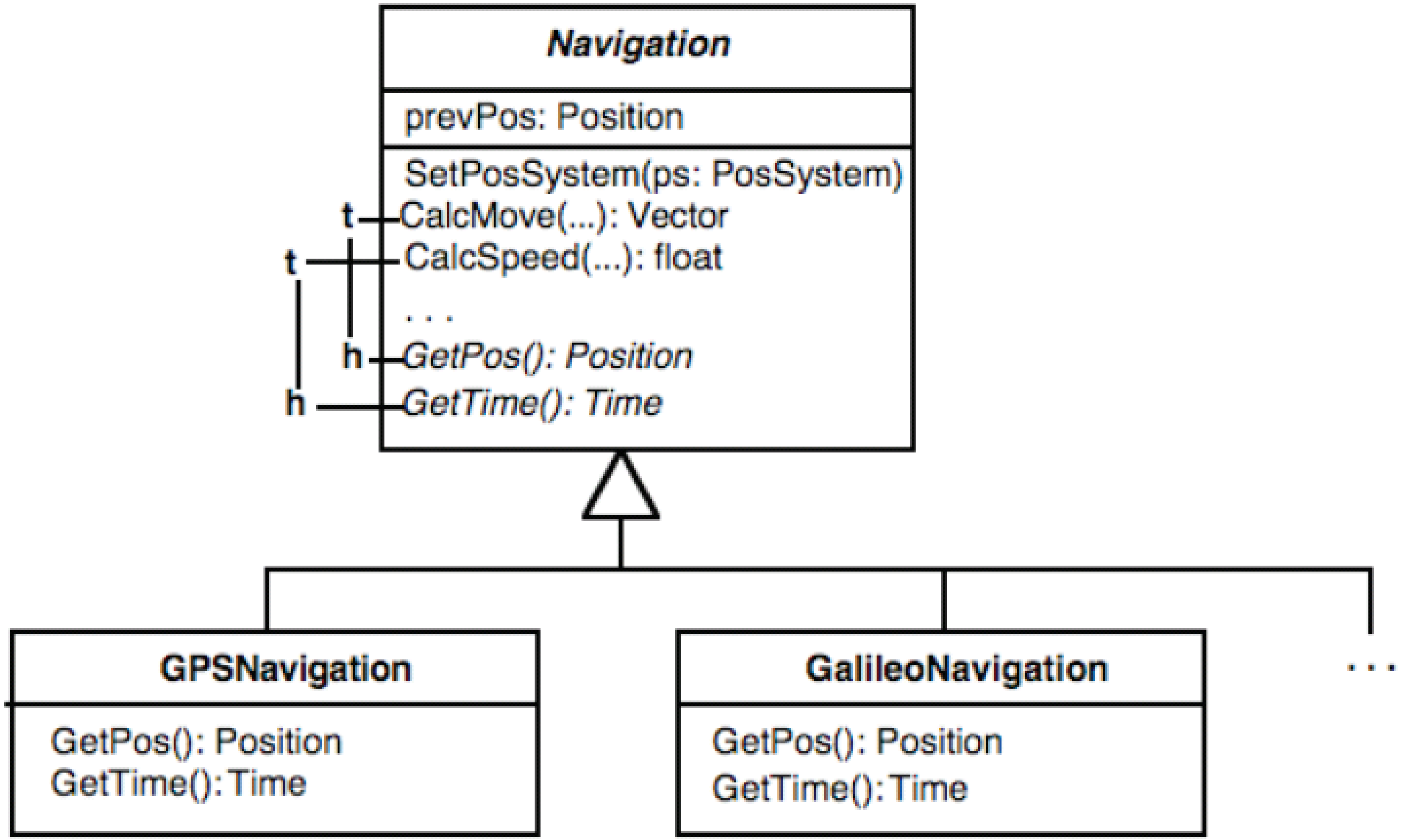


Adaptation by Overwriting the Hook Method H()

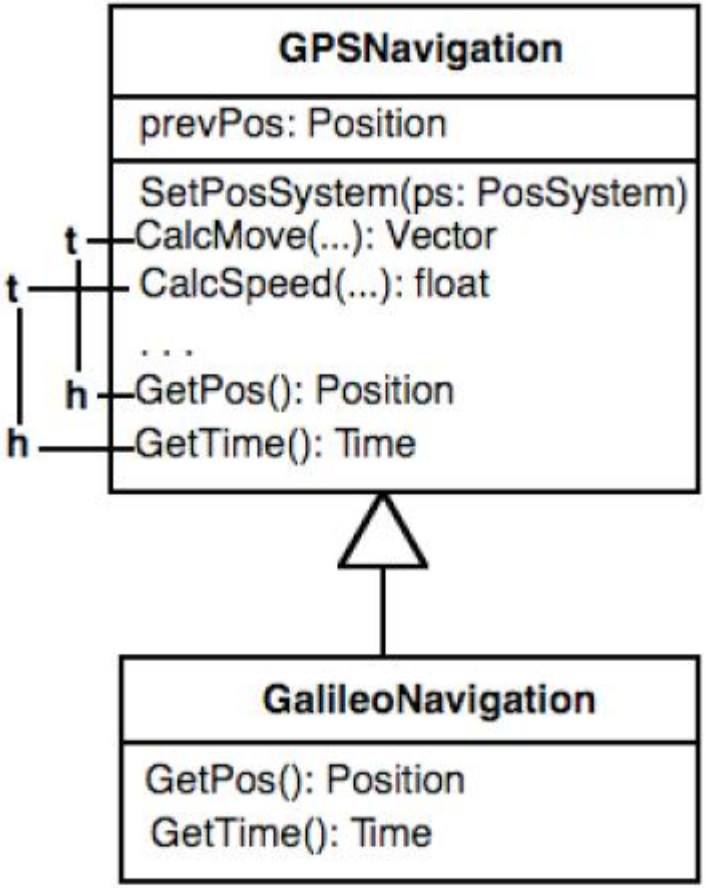


```
T() {
    . . .
    H();  // call H
    . . .
}
```

Application Example: Navigation System(I)



Application Example: Navigation System(II)



Problem: Galileo is not a specialization of GPS!

Summary *Hook Method*

+ Simplicity: For an adaptable behavior, one must plan only a hook method.

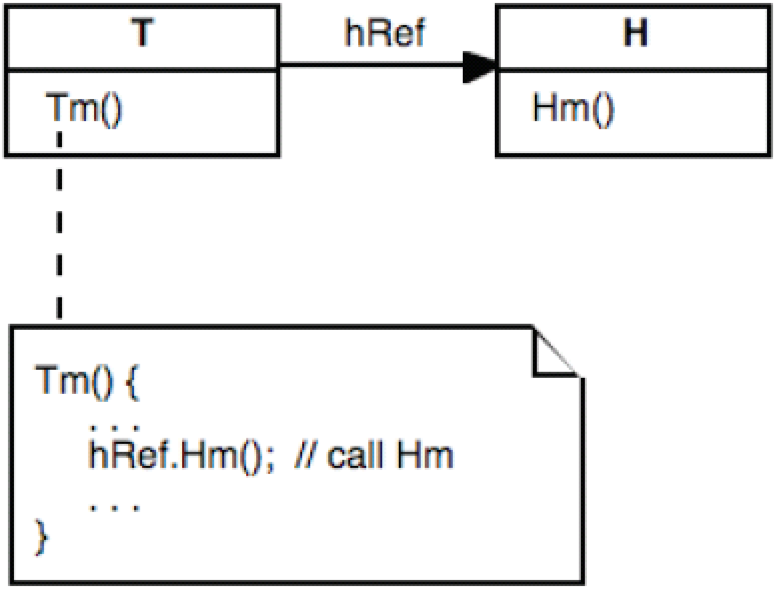
- Adaptability requires sub-classing and overwriting of the hook method.

In many cases, the hook method construction principle is sufficient to achieve the flexibility required for adaptation.

UNIVERSITÄT SALZBURG

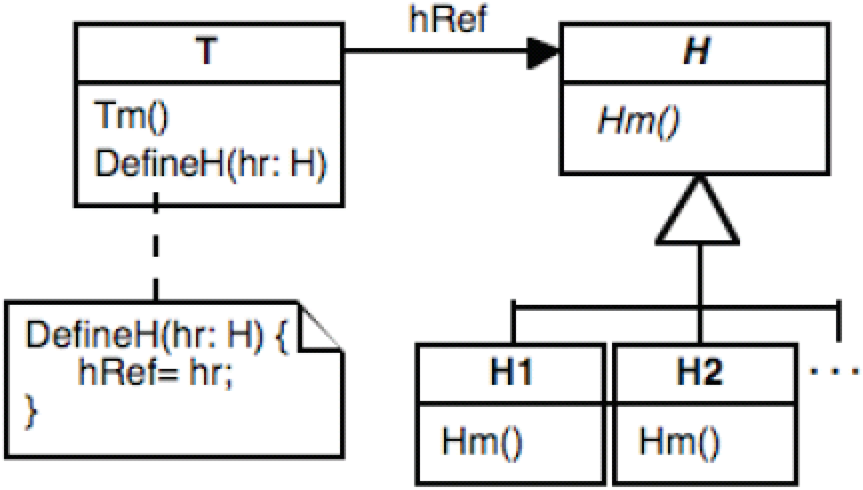
The Hook Object Construction Principle

Hook Object: Adaptation of T() by plugging in an H Object



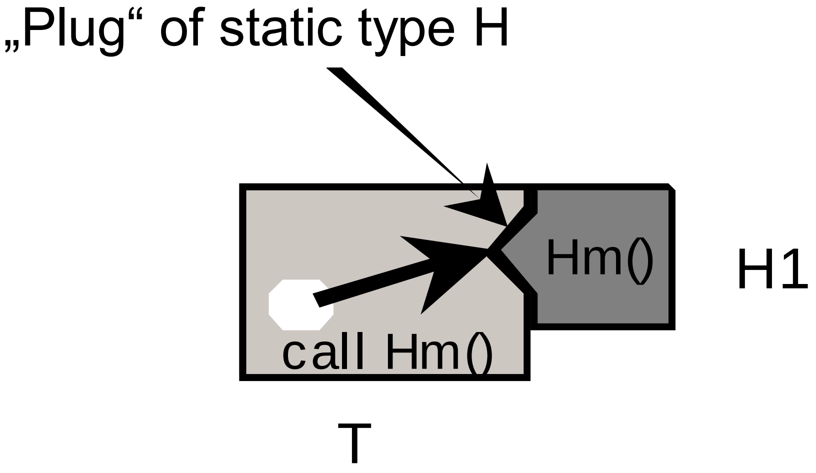
```
Tm() {
    . . .
    hRef.Hm();  // call Hm
    . . .
}
```

Adaptation by Composition (I)



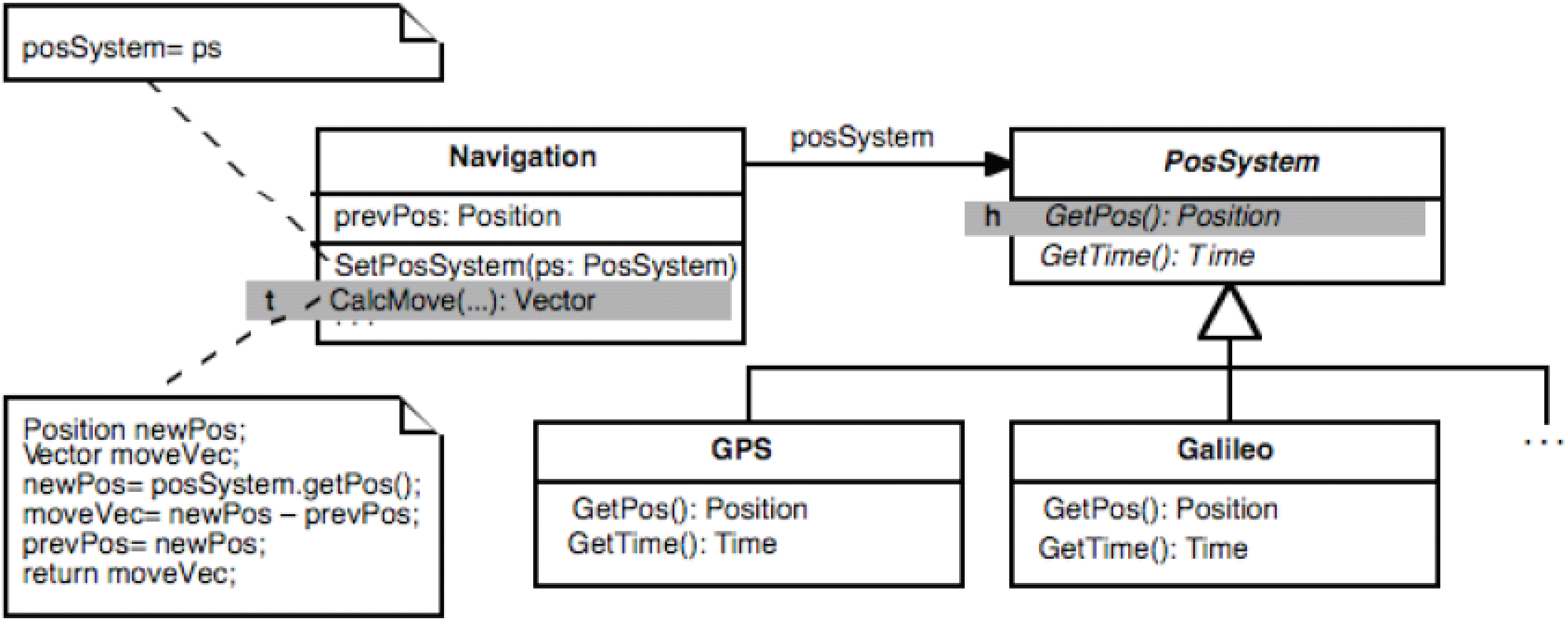
⇒ Adaptability at runtime

Adaptation by Composition (II)

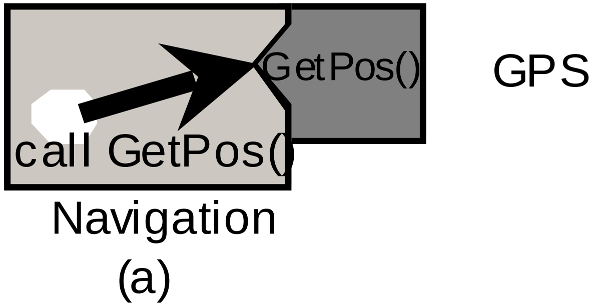


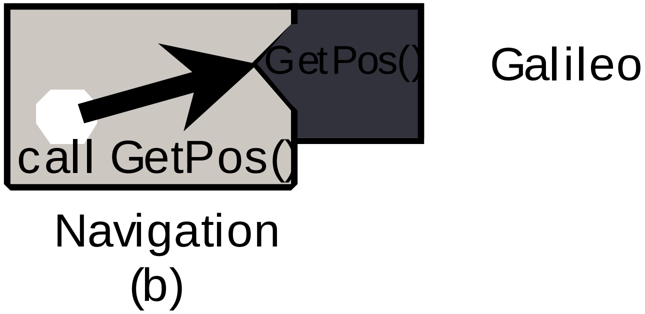
```
T sampleT= new T();
sampleT.DefineH(new H1());
```

Application Example: Navigation System(I)



Application Example: Navigation System(II)



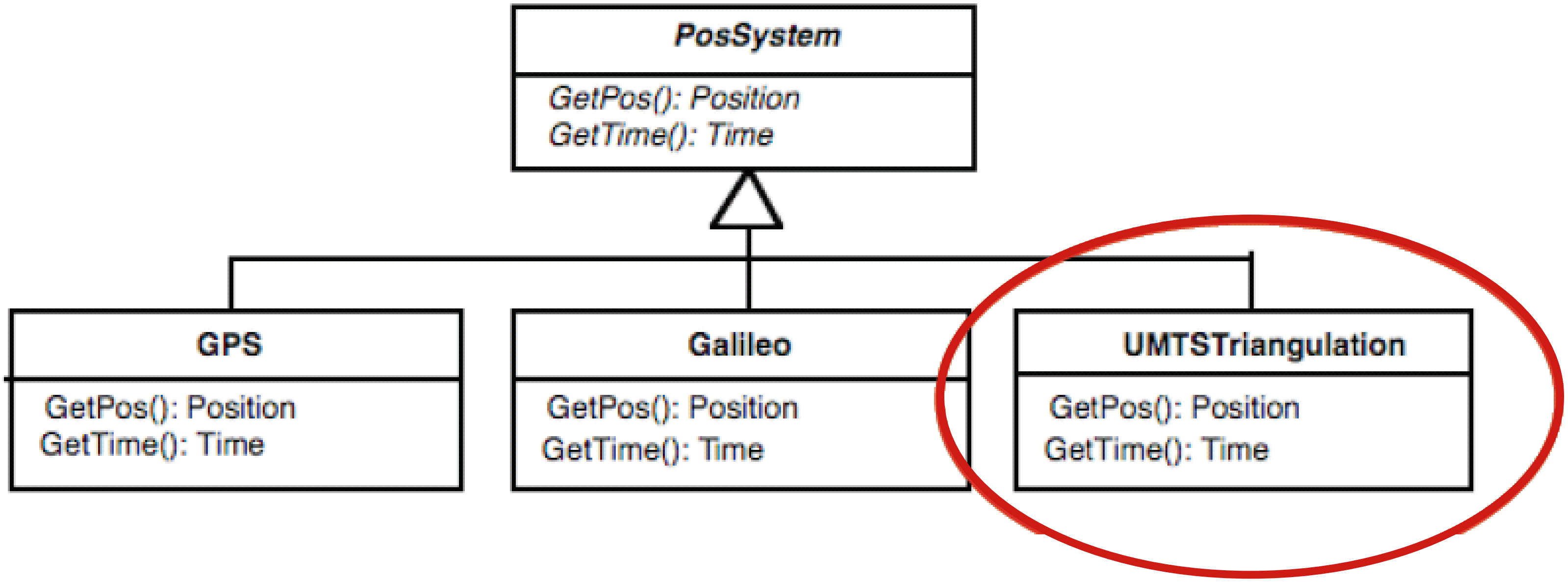


Composition for achieving a navigation system:

(a) GPS-based

(b) Galileo-based

Extension of the Pluggable Components at Runtime?

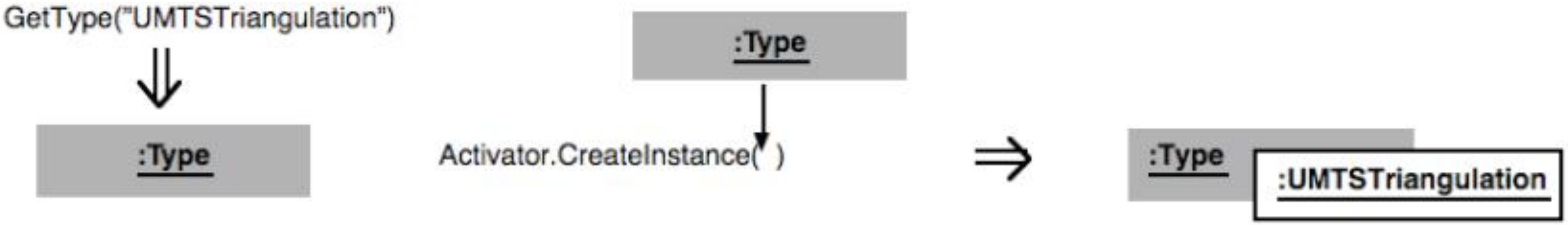


```
Navigation navigation= new Navigation(...);
String nameOfAddtlClass= “UMTSTriangulation“;
Object anObj= new nameOfAddtlClass;  // not possible
                                     // correct solution follows
navigation.SetPosSystem((PosSystem)anObj);
```

Using dynamic class loading in Java

```
Navigation navigation= new Navigation(...);

String nameOfAddtlClass= “UMTSTriangulation“;

ClassLoader classLoader = navigation.getClass.getClassLoader();

try {
    Class newPosSystCls = classLoader.loadClass(nameOfAddtlClass);

    PosSystem newPosSystObj = (PosSystem) newPosSystCls.newInstance();

    navigation.SetPosSystem(newPosSystObj );

     } catch (ClassNotFoundException e) { e.printStackTrace(); }
```

By Reflection in .NET/C#



```
Navigation navigation= new Navigation(...);
. . .
String nameOfAddtlClass= “UMTSTriangulation“;
Type typeOfAddtlClass= Type.GetType(nameOfAddtlClass);
Object anObj;
PosSystem posSys;

if (typeOfAddtlClass != null) {
   anObj= Activator.CreateInstance(typeOfAddtlClass);
    if (anObj != null && anObj is PosSystem)
        posSystem= (PosSystem) anObj;
    else  ...  // error handling
}
navigation.SetPosSystem(posSys);
```

Summary *Hook Object*

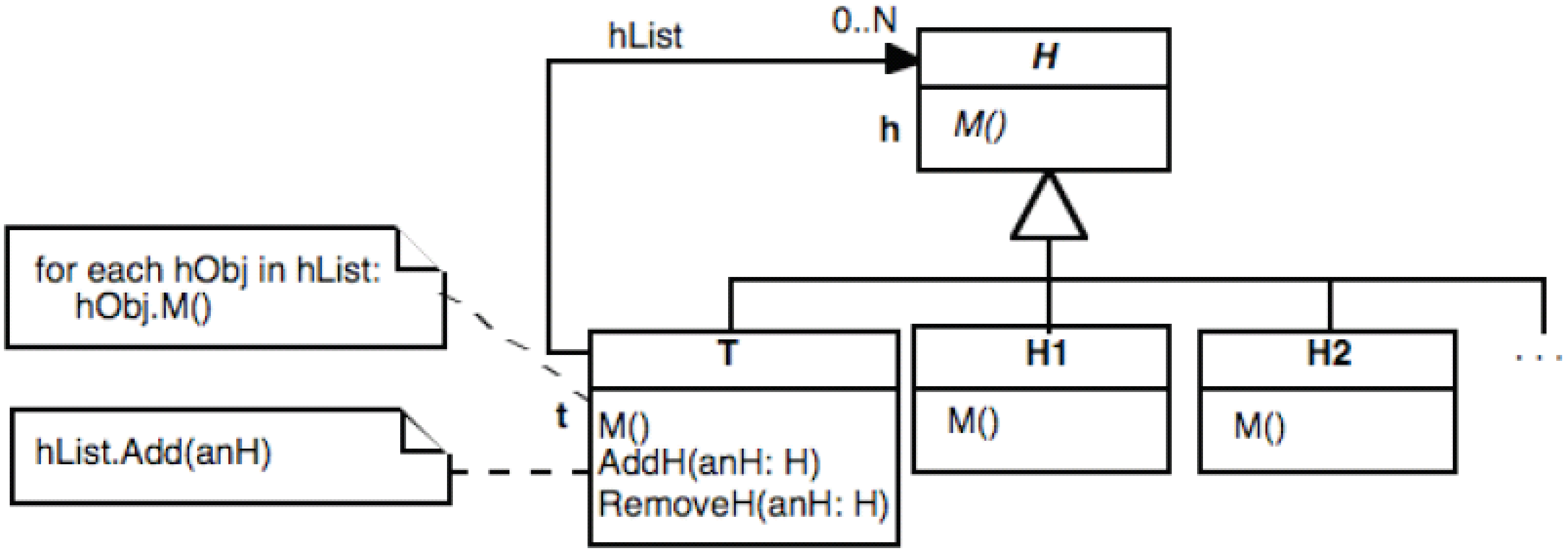
+ Simple configuration, also at runtime

- Higher complexity of design and implementation than in the hook method principle

The Composite Construction Principle

UNIVERSITÄT SALZBURG

Composite: A tree of objects can be used like an individual object

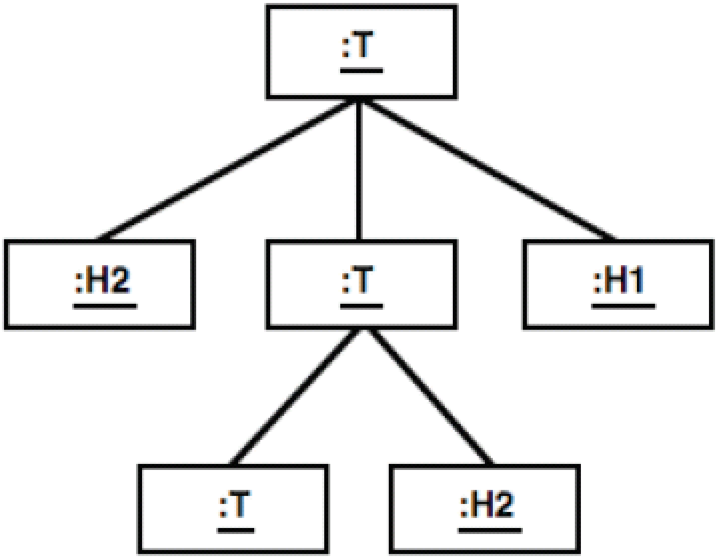


- The names of template and hook methods are the same
- References to H-objects are managed by AddH() and RemoveH()

UNIVERSITÄT SALZBURG

Example: Definition of an Object Hierarchy

```
T root= new T();
T subRoot= null;
root.AddH(new H2());
subRoot= new T();
root.AddH(subRoot);
root.AddH(new H1());
subRoot.AddH(new T());
subRoot.AddH(new H2());
```



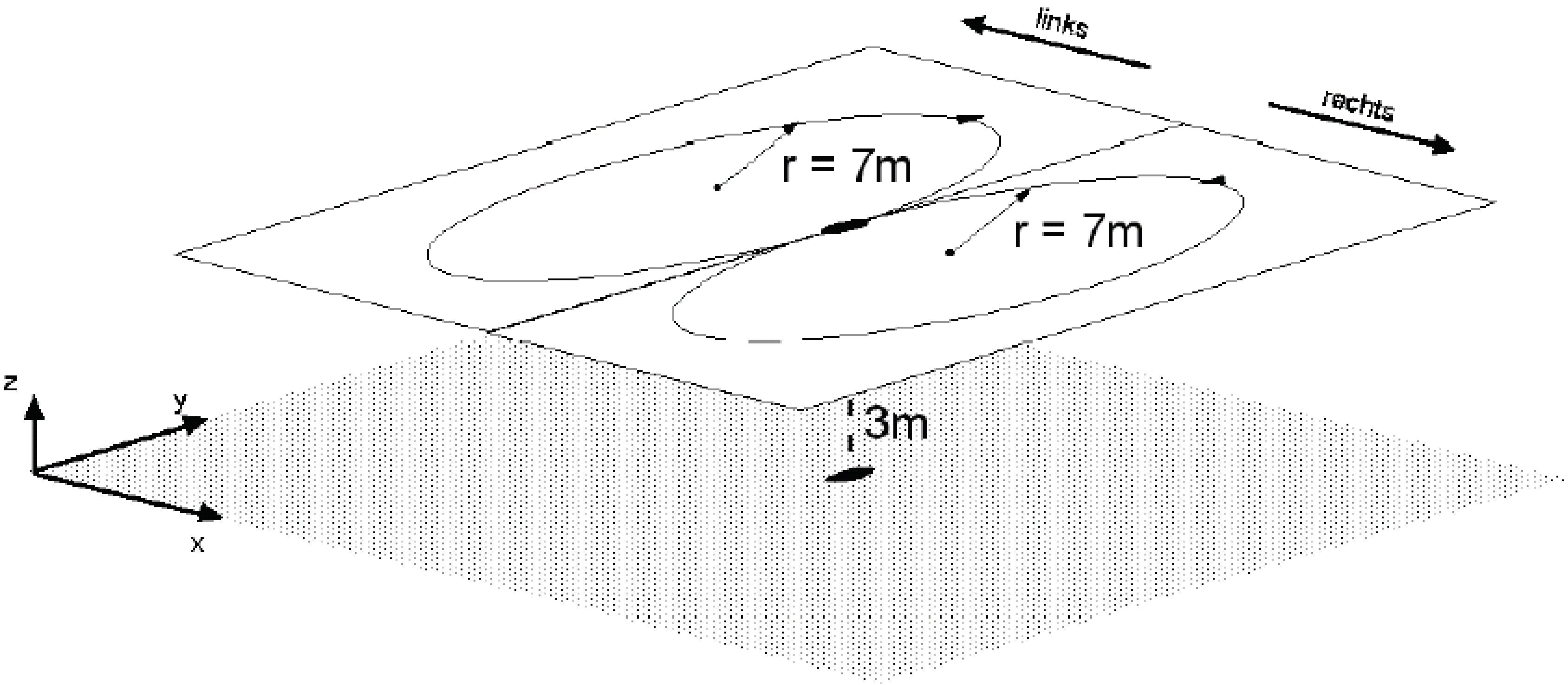
The object hierarchy can be used by the structure of the template method like an object

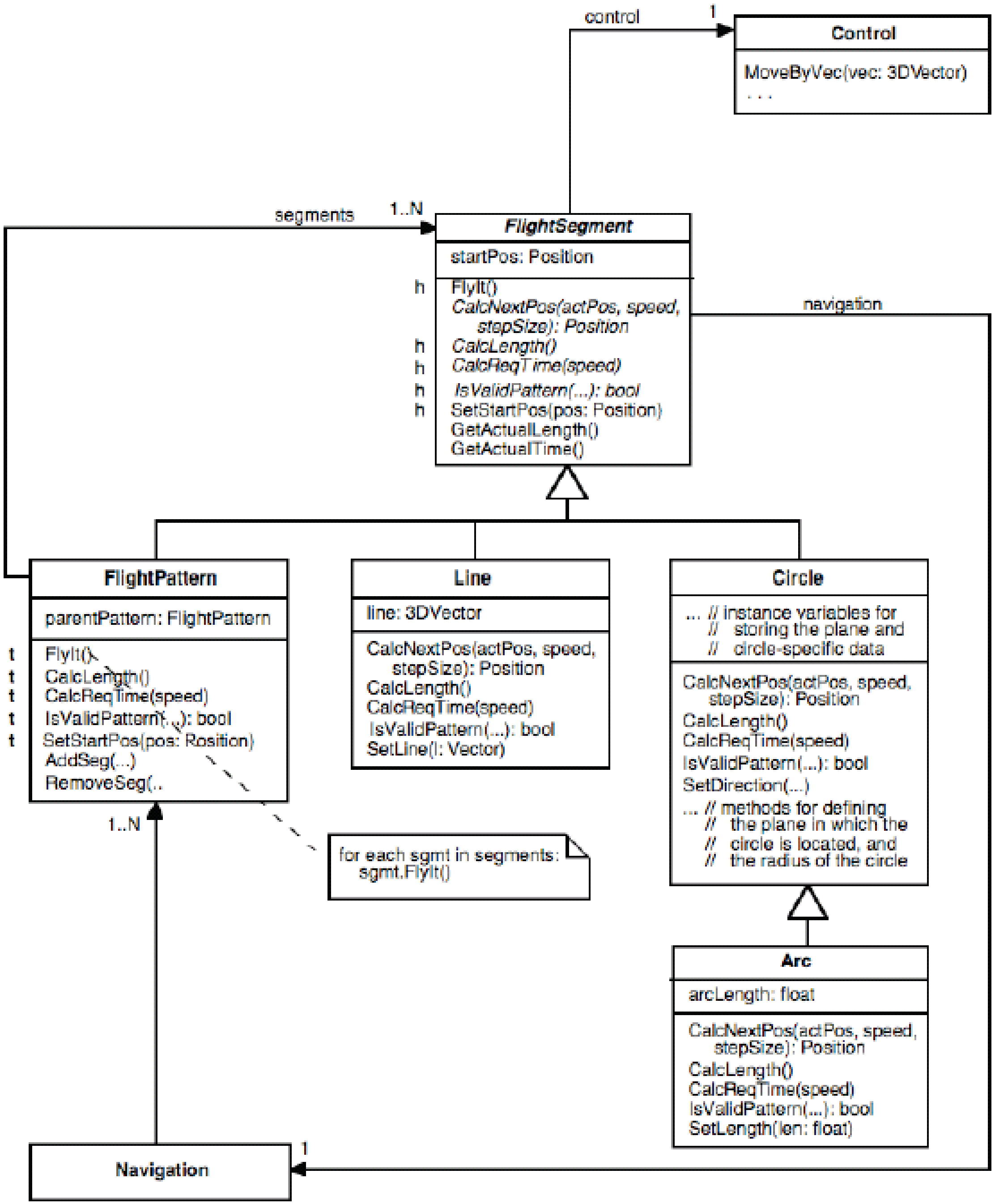
```
void M() {
     for each hObj in hList
         hObj.M();
  }
```

M () is not a recursive method, however it operates on a recursive data structure (tree).

UNIVERSITÄT SALZBURG

Example: Composition of an 8-flight Pattern From Segments





control 1

Control

MoveByVec(vec: 3DVector)
. . .

segments 1..N

FlightSegment

startPos: Position

h FlyIt()
CalcNextPos(actPos, speed, stepSize): Position
h *CalcLength()*
h *CalcReqTime(speed)*
h *IsValidPattern(...): bool*
h SetStartPos(pos: Position)
GetActualLength()
GetActualTime()

navigation

FlightPattern

parentPattern: FlightPattern

t FlyIt()
t CalcLength()
t CalcReqTime(speed)
t IsValidPattern(...): bool
t SetStartPos(pos: Position)
AddSeg(...)
RemoveSeg(..

1..N

for each sgmt in segments:
sgmt.FlyIt()

Line

line: 3DVector

CalcNextPos(actPos, speed, stepSize): Position
CalcLength()
CalcReqTime(speed)
IsValidPattern(...): bool
SetLine(l: Vector)

Circle

... // instance variables for
// storing the plane and
// circle-specific data

CalcNextPos(actPos, speed, stepSize): Position
CalcLength()
CalcReqTime(speed)
IsValidPattern(...): bool
SetDirection(...)
... // methods for defining
// the plane in which the
// circle is located, and
// the radius of the circle

Arc

arcLength: float

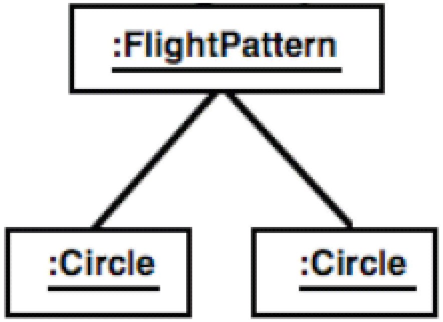
CalcNextPos(actPos, speed, stepSize): Position
CalcLength()
CalcReqTime(speed)
IsValidPattern(...): bool
SetLength(len: float)

Navigation

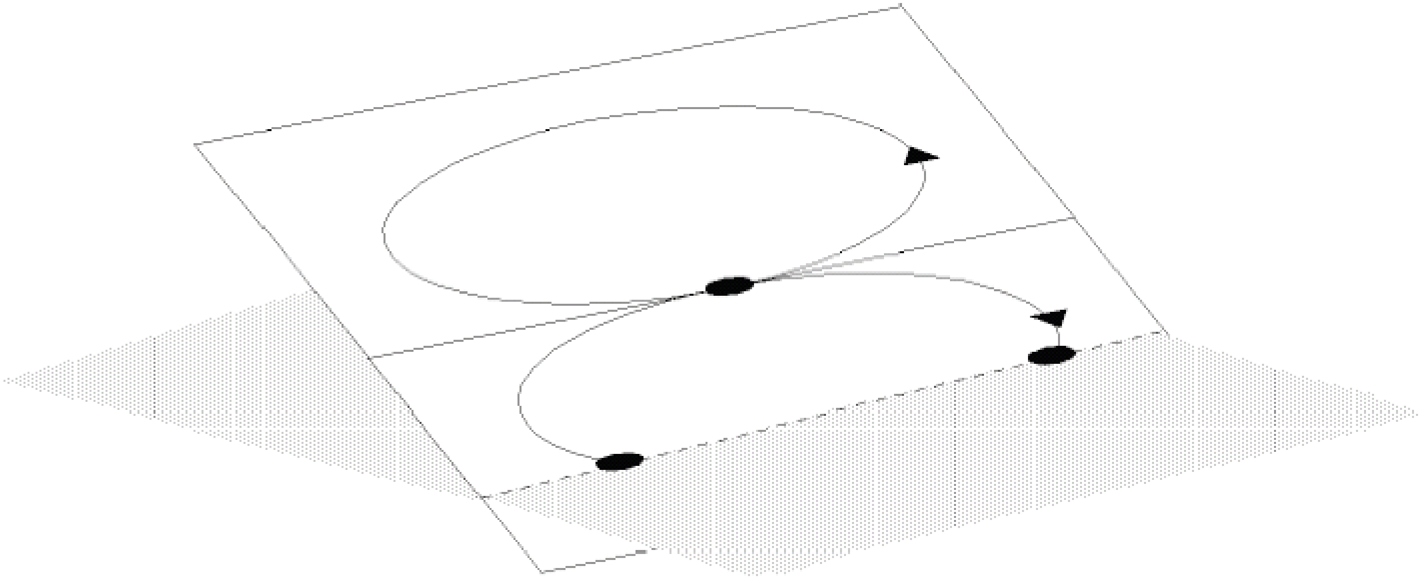
1

The 8-loop

```
FlightPattern loop= new FlightPattern();
loop.SetStartPos(new Position(gL, gB) + new Position(0, 0, 3));
loop.AddSeg(new Circle (horizontalPlane, 7, right));  // radius: 7 m; right dir.
loop.AddSeg(new Circle (horizontalPlane, 7, left));  // radius: 7 m; left dir.
```



IsValidPattern() cheks whether a flight pattern leads to a ground contact

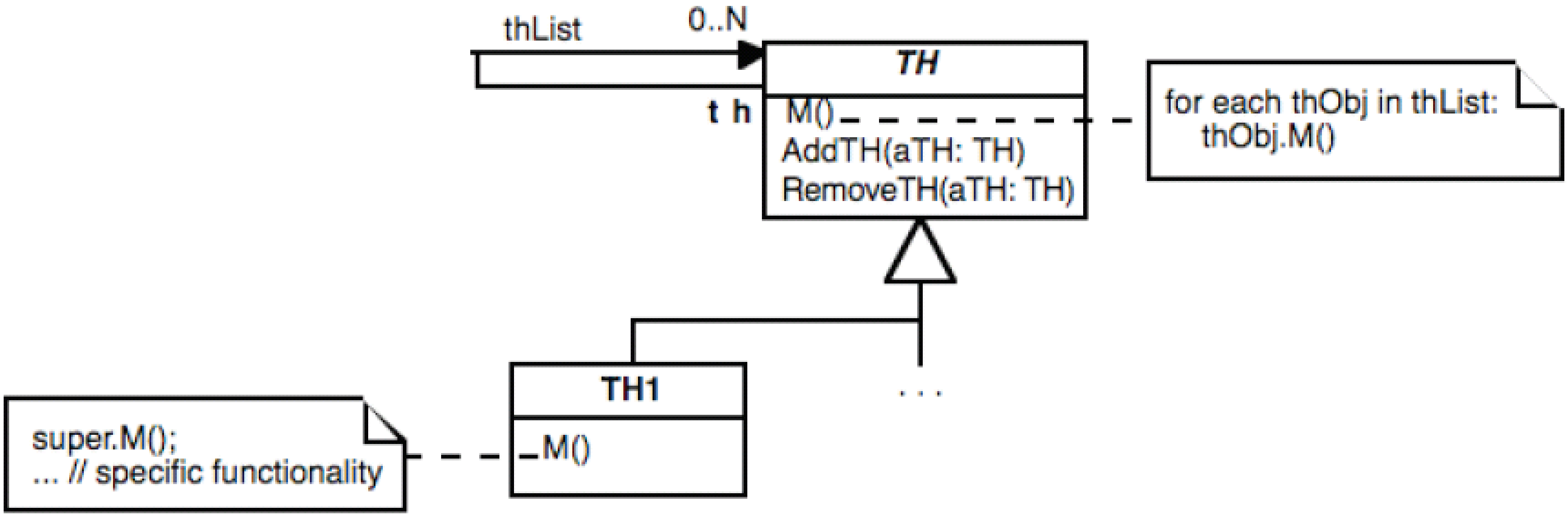


- IsValidPattern () is implemented in FlightPattern in accordance with the Composite template method
- Similarly: FlyIt (), CalcLength (), CalcReqTime ()
- FlyIt () is already implemented by using

FlightSegment - > CalcNextPos ()

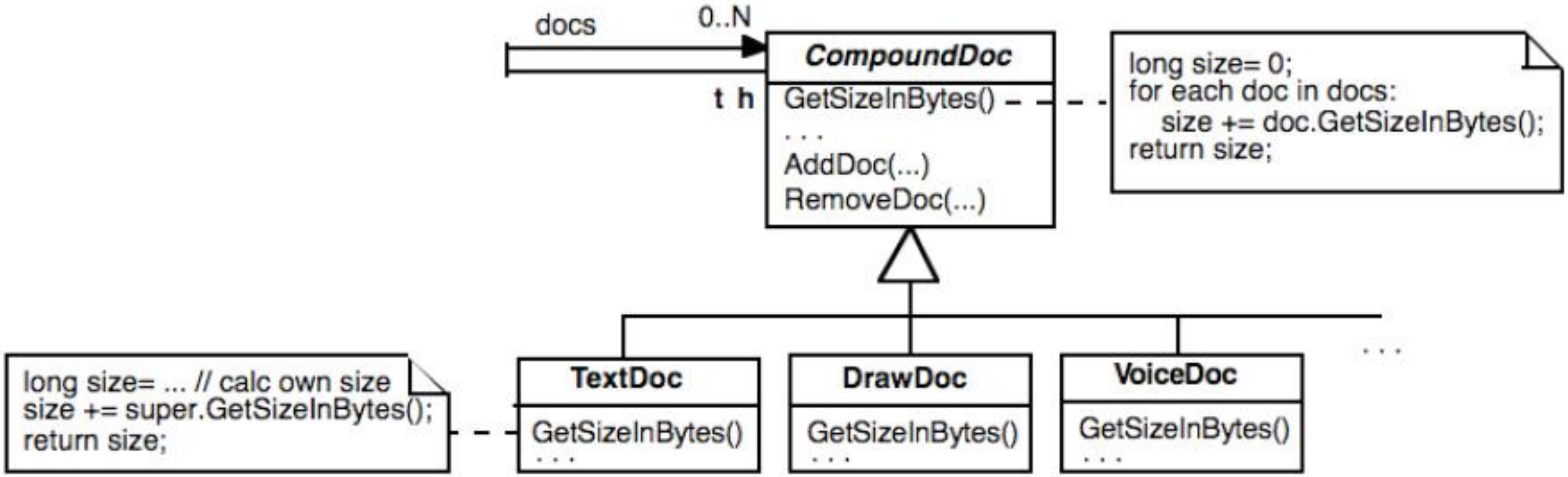
Composite Variant: Administration and Functionality in One Class

- T and H class merged
- Semantics of the composition changes
- The fundamental characteristic to be able to define an object hierarchy remains



UNIVERSITÄT SALZBURG

Example: Complex Documents



CompoundDoc

GetSizeInBytes()
. . .
AddDoc(...)
RemoveDoc(...)

```
long size= 0;
for each doc in docs:
   size += doc.GetSizeInBytes();
return size;
```

TextDoc

GetSizeInBytes()
. . .

DrawDoc

GetSizeInBytes()
. . .

VoiceDoc

GetSizeInBytes()
. . .

```
long size= ... // calc own size
size += super.GetSizeInBytes();
return size;
```

A document that comprises text and different other documents like drawings, audio or video clips, is responsible for the administration of the contained documents and offers additional functionality for editing the embedded documents.

Summary *Composite*

+ Simple formation of flexible object hierarchies
+ New elements (subclasses of the hook class) without change of the template class
- Complexity of interactions between objects arranged in the hierarchy, in order to accomplish the automatic iteration over the tree hierarchy.

Object hierarchies occur very frequently and in many ranges of application, e.g. in window–grouped GUI elements, parts lists, workflows.

UNIVERSITÄT SALZBURG