Component Diagrams
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
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
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
Contents

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

- Configuration parameters are placed in configuration files.
- Configuration parameters correspond to persistent, global (= static) variables.
Example

Konfigurationsdatei

:ConfigInterpreter

:Converter

roundingParam: float

0.01

Rundungsgenauigkeit 0.01

CurrencyConversion

Legende:

:ObjectName

Objekt

externe Datei

Softwarekomponente
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
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)

```c
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)
**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

„Stecker“ PosSystem

call GetPos()

GetPos()
Abstract Coupling by Abstract Classes

Navigation system example:

```
posSystem = ps

Navigation
prevPos: Position
SetPosSystem(ps: PosSystem)
CalcMove(...) : Vector

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

GPS
GetPos(): Position
GetTime(): Time

Galileo
GetPos(): Position
GetTime(): Time
```
Alternative: Interfaces
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

```java
M1() {
    M2(); // call M2
    ...
}
```
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
Hook Method
Construction
Principle
Hook Method: Adaptation of \( T() \) by overwriting of \( H() \)
Adaptation by Overwriting the Hook Method 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.
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)

```java
posSystem = ps

Navigation
prevPos: Position
SetPosSystem(ps: PosSystem)
CalcMove(...): Vector

Position newPos; Vector moveVec;
newPos = posSystem.getPos();
moveVec = newPos - prevPos;
prevPos = newPos;
return moveVec;

PosSystem
h
GetPos(): Position
GetTime(): Time

GPS
GetPos(): Position
GetTime(): Time

Galileo
GetPos(): Position
GetTime(): Time
```
Application Example: Navigation System (II)

Composition for achieving a navigation system:
(a) GPS-based
(b) Galileo-based
Extension of the Pluggable Components at Runtime?

```java
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(); }
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
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()
Example: Definition of an Object Hierarchy

```java
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

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

M() is not a recursive method, however it operates on a recursive data structure (tree).
Example: Composition of an 8-flight Pattern From Segments
The 8-loop

```java
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() checks 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
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.