Component Diagrams
Components

- Autonomous, encapsulated units within a system or subsystem that provide one or more interfaces
- Correspond to modules in module-oriented languages
- In C++: Reproduction of modules through .h, .c files
- In Smalltalk: Groups of classes, no modules
- In Oberon and Java: Modularity supported directly by the language

UML representation:

- [Diagram showing UML 2.0 and 1.4 representations of components]
Ports, Interfaces and Connectors

- **Ports:** interaction points
- **Interfaces:**
  - Provided
  - Required
- **Connectors:**
  - Assembly
  - Delegation
Component Diagram

- **Purpose:** show the structural relationships between constituents of a system
- **Natural format for system architects to start modeling a solution**
- **Useful communication means for various groups, not just in the implementation but also in the stakeholder circles**
  - Stakeholders have an early understanding of the system
  - Developers can start drafting the work tasks
  - System administrators can do early deployment plans
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

Client → Middleware → Legacy System

Legacy System
Web Shop: Topology

Client PC
  
<table>
<thead>
<tr>
<th>Browser</th>
</tr>
</thead>
</table>

Database Server

| DBMS |

Web Server

| :statische Html Files |

| :JSP Files |

| WebShop.jar |

TCP/IP

Stefan Resmerita, WS2015
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

CurrencyConversion

:ConfigInterpreter

:Converter

roundingParam: float

0.01

Rundungsgenauigkeit

Softwarekomponente

Legende:

:ObjectName

Objekt

externe Datei

Stefan Resmerita, WS2015
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).
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
Navigationskomponente

„Stecker“ PosSystem

call GetPos()

GPS- Komponente
Galileo- Komponente

Navigationskomponente
Abstract Coupling by Abstract Classes

Navigation system example:

```
posSystem = ps

Navigation

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

PosSystem
GetPos(): Position
GetTime(): Time

GPS
GetPos(): Position
GetTime(): Time

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

```c
Position newPos;
Vector moveVec;
newPos = posSystem.GetPos();
movesVec = newPos - prevPos;
prevPos = newPos;
return moveVec;
```
Alternative: Interfaces

```
interface I {
    M1();
    M2();
}

class A implements 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

```java
A

M1()
M2()
...

M1() {
    M2(); // call M2
    ...
}
```
Template and Hook Methods in Different Classes

```cpp
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)

```
Navigation
prevPos: Position
SetPosSystem(ps: PosSystem)
CalcMove(...): Vector
CalcSpeed(...): float
...
GetPos(): Position
GetTime(): Time

GPSNavigation
GetPos(): Position
GetTime(): Time

GalileoNavigation
GetPos(): Position
GetTime(): Time
```
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
Adaptation by Object Composition (I)

⇒ Adaptability at runtime
Adaptation by Object 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

```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 collection 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

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

```c
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

FlightPattern loop = new FlightPattern();
loop.SetStartPos(new Position(gL, gB) + new Position(0, 0, 3));
loop.AddSeg(new Circle (horizontalPlane, 7, right));
loop.AddSeg(new Circle (horizontalPlane, 7, left));
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

```
super.M();
... // specific functionality

for each thObj in thList:
thObj.M()
```

Diagram:
```
TH

thList 0..N

M()
AddTH(aTH: TH)
RemoveTH(aTH: TH)

TH1

M()

... // specific functionality
```
Example: Complex Documents

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.