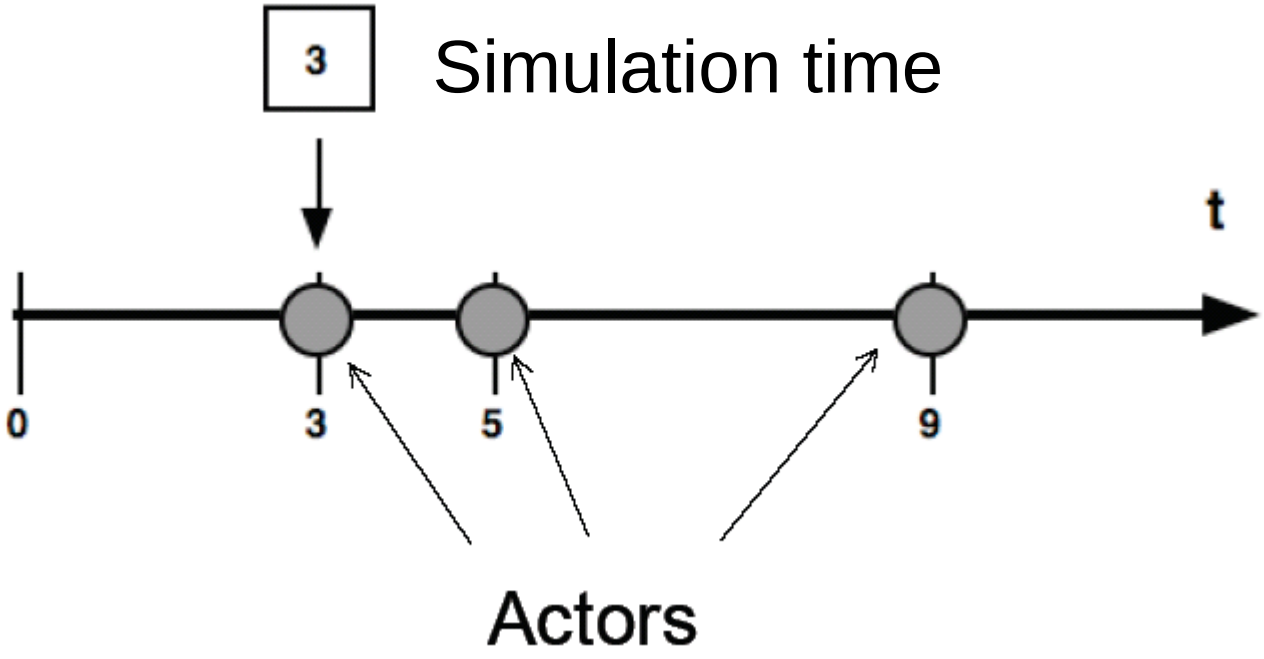
Example: Simulation of Discrete Events

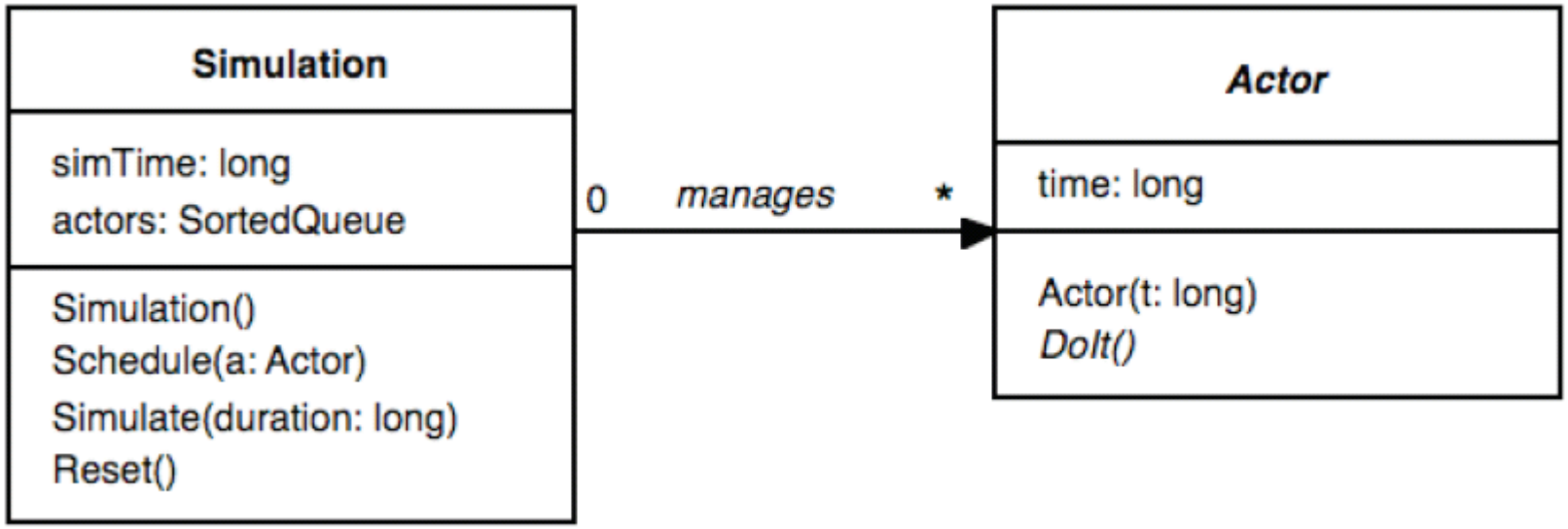
UNIVERSITÄT SALZBURG

Discrete Events on a Time Axis



UNIVERSITÄT SALZBURG

Framework for discrete event simulation



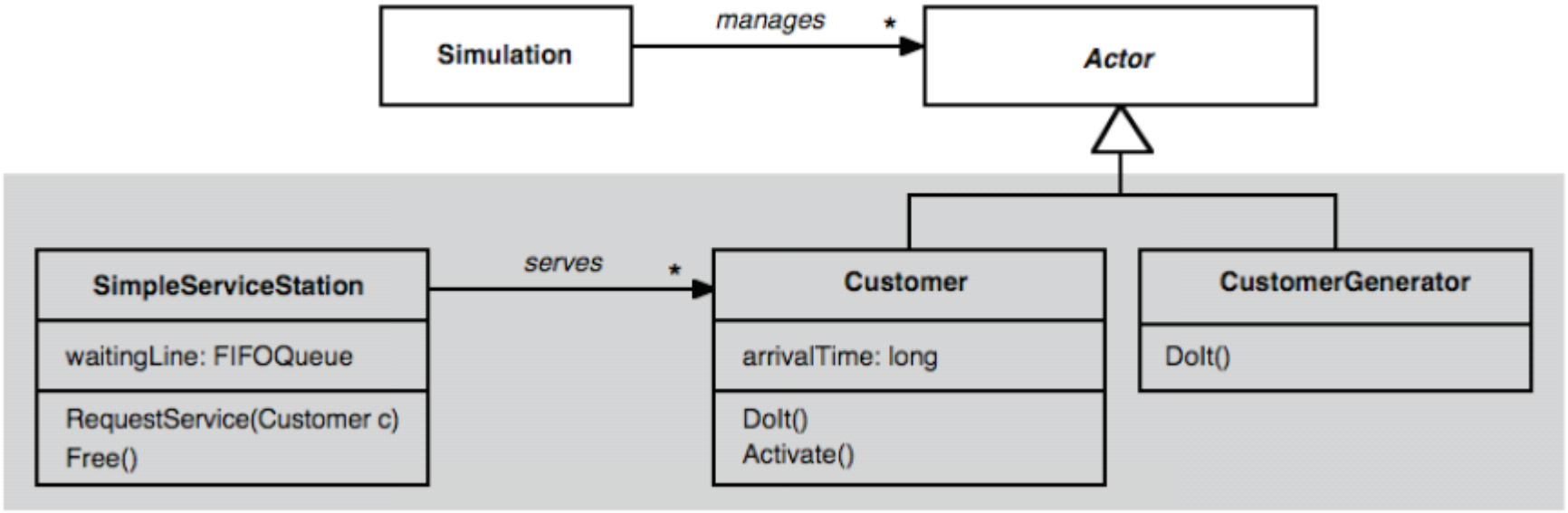
UNIVERSITÄT SALZBURG

C# Implementation of Simulate()

```
public void Simulate(long duration) {
        long endOfSimulation= simTime + duration;
        do {
          if (actors.Count() != 0) {
                Actor actor= (Actor) actors.Dequeue();
                simTime = actor.time;
                actor.DoIt();
          } else      // no more actors enqueued
                break;      // exit loop
        } while (simTime <= endOfSimulation);
    }
```

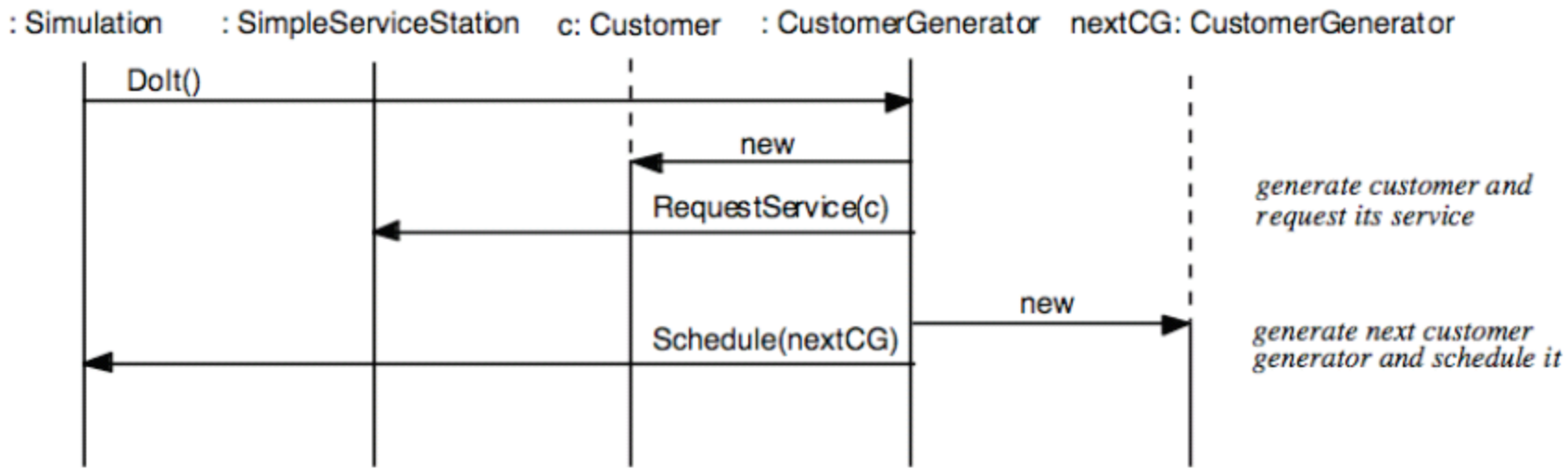
UNIVERSITÄT SALZBURG

Classes for the simulation of a simple bank service counter



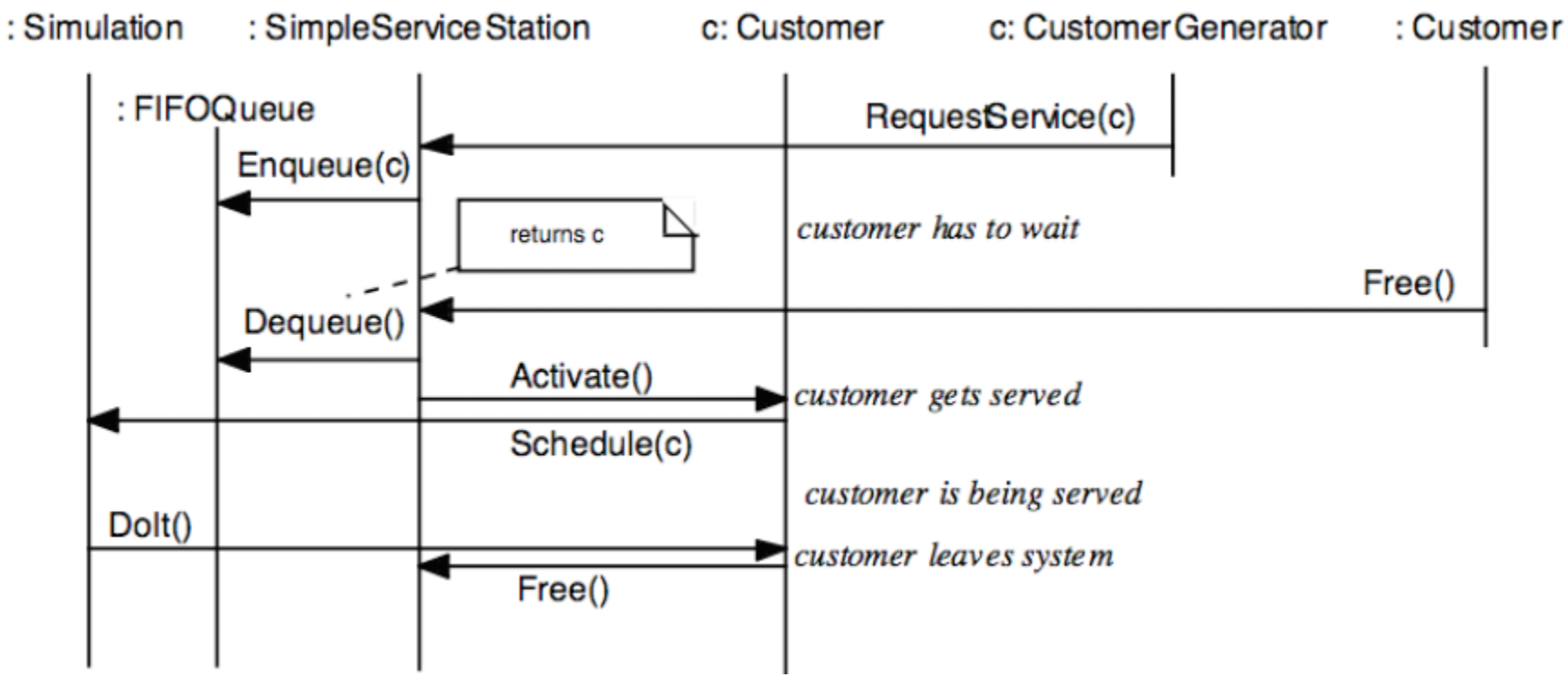
UNIVERSITÄT SALZBURG

The DoIt() Method of Class CustomerGenerator

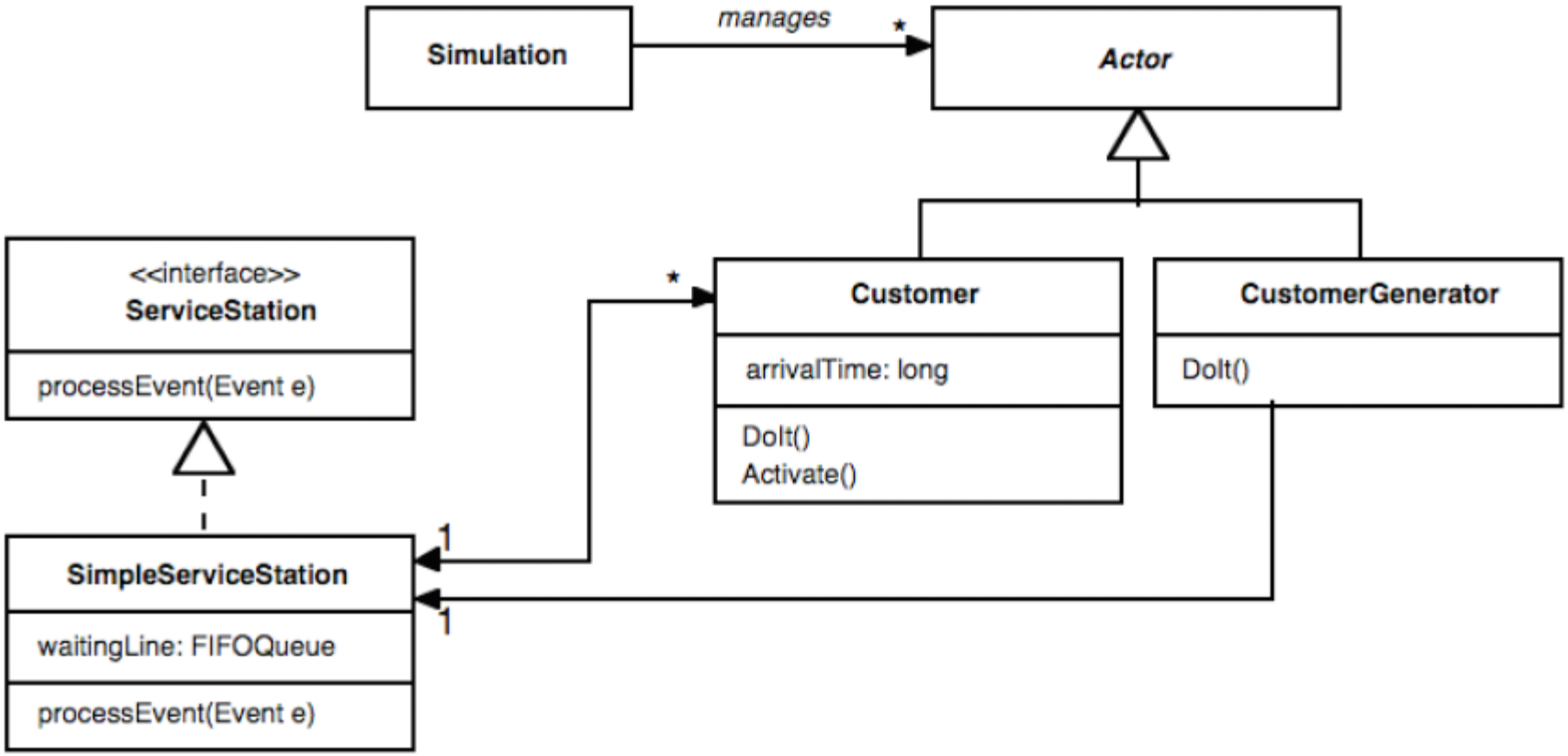


UNIVERSITÄT SALZBURG

Lodging a customer in the queue



Decrease of the coupling between service station and the actors



UNIVERSITÄT SALZBURG

Software Architecture Analysis Method (SAAM)

UNIVERSITÄT SALZBURG

Architecture Analysis: what for ?

- When building a software system, the analysis of different architecture options (also called candidate architectures) permits to estimate fulfillment of quality attributes. Different candidate architectures should be evaluated in the design phase and be accepted or rejected based on reliable judgements.
- In addition, an architecture analysis is meaningful if an existing software system is to be acquired or transferred. Only by a sufficient architecture analysis it is possible to estimate the product quality, in particular regarding the maintenance, adaptability and expandability under the given general conditions.

UNIVERSITÄT SALZBURG

Selection of relevant criteria in SAAM (I)

- We consider the quality criterion **adaptability** as an example. A software system can be easily adaptable in certain aspects (for example structure and configuration of the user interface), while being hardly adaptable in other aspects (for example in supported data formats and conversions).

- The requirements related to a certain quality criterion must be defined therefore in relation to a certain context.

UNIVERSITÄT SALZBURG

Selection of relevant criteria in SAAM (II)

- SAAM specifies context through **scenarios**. For example, a scenario describes that a user of the software is able to configure the user interface. The user mentioned in the scenario is an example of a concerned party (***stakeholder*** in the SAAM terminology), e.g., a person who has purchased the software system. A **scenario** describes in terms of catchwords the interaction of a stakeholder with the software system.
- Scenarios are the premises of a useful software architecture analysis.

UNIVERSITÄT SALZBURG

SAAM steps

(1) **Identify and assemble stakeholders**

(2) **Develop and prioritize scenarios** (functionality scenarios, change and development scenarios, etc.)

(3) Describe candidate architectures

(4) **Classify scenarios as direct or indirect**

(5) Evaluate indirect scenarios by an analysis of each architecture, to obtain an **estimation of the coupling of the components** of the software architecture

(6) Evaluate the interactions between indirect scenarios by an analysis of the architecture, to obtain an **estimation of the cohesion of the components** of the software architecture

(7) Generate an overall evaluation

UNIVERSITÄT SALZBURG

SAAM – possible stakeholders (I)

Stakeholder	Interest
Customer	Schedule and budget Usefulness of system Meeting market expectations
End user	Functionality Usability Robustness
Developer Maintainer Integrator Application builder	Modularity (Coupling, Cohesion) Documentation System transparency / Readability (e.g., ability to locate places of change)

UNIVERSITÄT SALZBURG

SAAM – possible stakeholders (II)

System administrator	Ease in finding sources of operational problems
Network administrator	Network performance Throughput predictability
Tester	Modularity (Coupling, Cohesion) Consistent error treatment Documentation System transparency / Readability
Representative of application area	Interoperability with other systems

UNIVERSITÄT SALZBURG

Description and prioritization of scenarios

- Scenarios are suggested by stakeholders and must be representative for future requirements, changes and extensions.
- The scenarios should describe all the relevant aspects of using the system and should refer in particular to aspects of functionality, development and change.
- Approx. 10 - 20 scenarios
- At the end: prioritization by importance

UNIVERSITÄT SALZBURG

Description of candidate architectures

- Practice has shown that a simple description of the data connections (which components exchange which information) and the control connections (one component enabling another component to perform its function) is usually sufficient for a static representation of architecture.
- In addition to the static description, runtime behaviors are outlined for example by UML interaction diagrams or by natural language.
- For scenarios that concern changes it may be necessary to include source text fragments.

UNIVERSITÄT SALZBURG

Direct and indirect scenarios

- Direct scenarios are those that can be executed by the system without modification.
- Indirect scenarios are those that require modifications to the system. In this case, it is necessary to indicate also the estimated expenditure for implementing the modifications (in person-days /-months or -years).
- The ensuing SAAM steps regard only the indirect scenarios. An indirect scenario can affect one or more components by the incurred changes of architecture.

UNIVERSITÄT SALZBURG

Evaluation of indirect scenarios

- For each indirect scenario, the number of components affected by changes is indicated.
- This is the measure for coupling in the software architecture.
- Also the cost of performing the changes is estimated.

UNIVERSITÄT SALZBURG

Evaluation of interactions between indirect scenarios

- Two indirect scenarios interact if they require a change of the same component.
- The analysis of scenario interactions points out which components cover several aspects and exhibit thus a small cohesion.
  - High interaction among scenarios that are fundamentally different corresponds to low cohesion and suggests high structural complexity.
  - High interaction among fundamentally similar scenarios signals high cohesion.

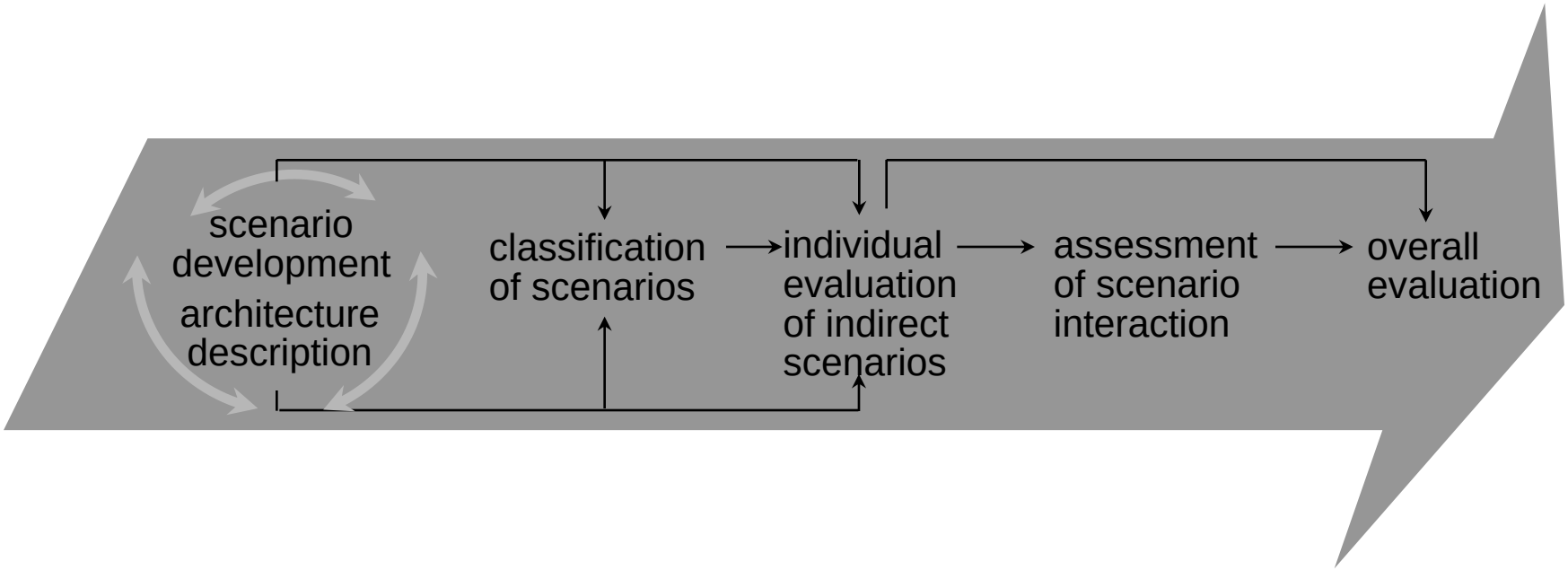
UNIVERSITÄT SALZBURG

Overall Evaluation

- The last step serves in particular to compare the different architecture candidates.
- A weight should be assigned to each scenario and the scenario interactions in terms of their relative importance, and the weighting should be used to determine an overall ranking of the candidate architectures.
- The process of performing a SAAM analysis can be used to gain a more complete understanding of the competing architectures; this understanding, rather than just a scenario-based table, is useful for performing comparative analysis.

UNIVERSITÄT SALZBURG

Interaction of SAAM Steps



UNIVERSITÄT SALZBURG

Example of SAAM application

UNIVERSITÄT SALZBURG

Revision Control System WRCS (I)

- A project in WRCS is a group of related files, which result together in a product if they are connected accordingly:
  - Source text files are those that are translated to an executable program,
  - Text documents of a book or
  - Digitized audio and video data for an advertising spot, etc.

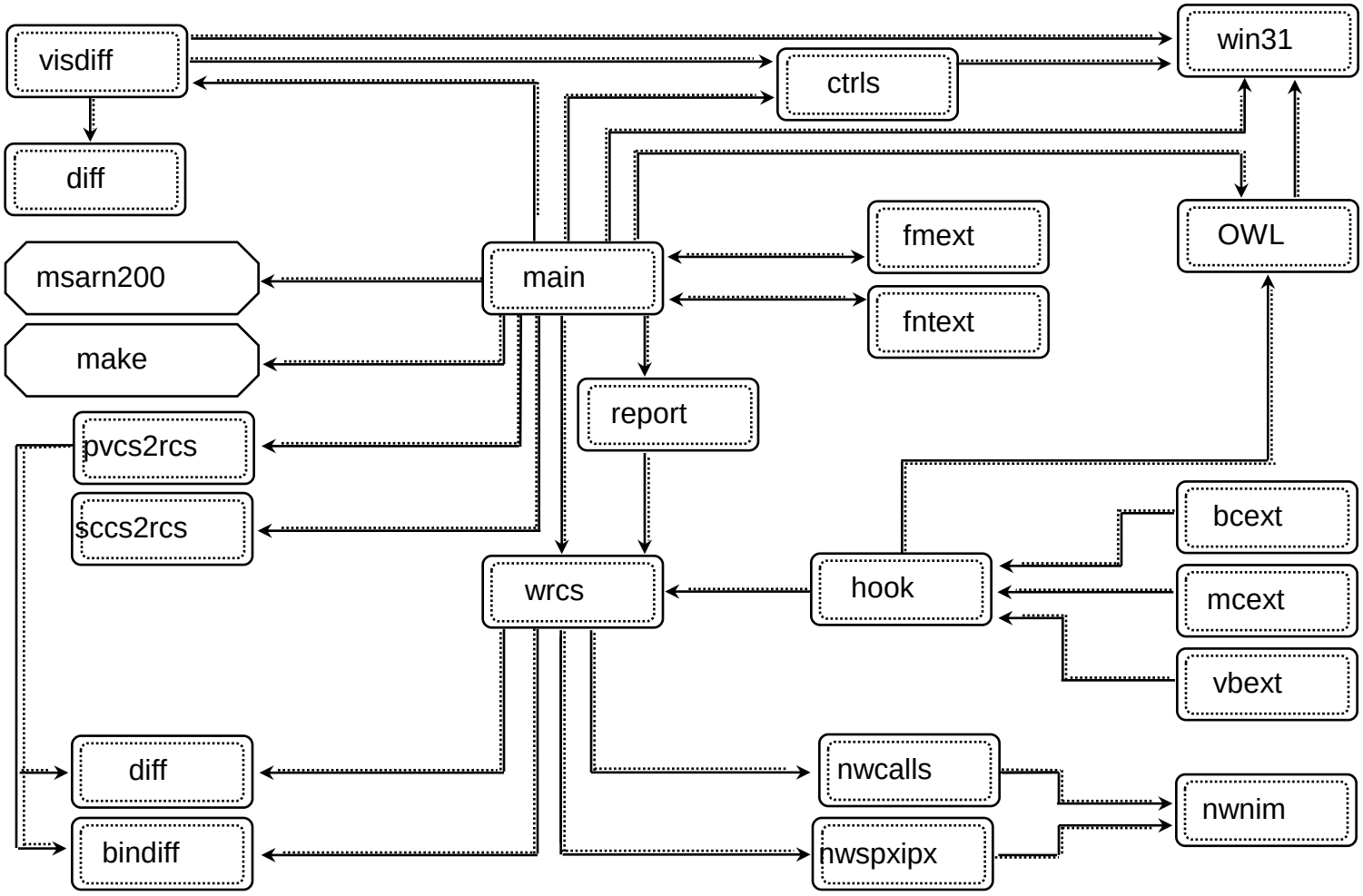
UNIVERSITÄT SALZBURG

Revision Control System WRCS (II)

- With WRCS one can
  - Trace changes of files
  - Define archives
  - Check files in and out
  - Create releases to be produced
  - Restore old versions
- The WRCS has been integrated with different development environments. The available functionality can be implemented by the respective development environments or via the WRCS's graphical user interface.

UNIVERSITÄT SALZBURG

Architectural Representation of WRCS



UNIVERSITÄT SALZBURG

Scenarios and their weights

Stakeholder	Scenario	Importance
End user	compare binary file representations	1
	configure the product's toolbar	3
Developer (Maintainer)	Make minor modifications to the user interface	6
	Port to another operating system	4
Administrator	Change access permissions for a project	5
	Integrate with a new development environment	2

UNIVERSITÄT SALZBURG

Classification in direct and indirect scenarios

Stakeholder	Scenario	Classification
End user	compare binary file representations	indirect
	configure the product's toolbar	direct
Developer (Maintainer)	Make minor modifications to the user interface	indirect
	Port to another operating system	indirect
Administrator	Change access permissions for a project	direct
	Integrate with a new development environment	indirect

UNIVERSITÄT SALZBURG

Number of components that must be changed by an indirect scenario

Indirect scenario	Number of components to change
Compare binary file representations	2
Configure the product's toolbar	3
Port to another operating system	3+
Integrate with a new development environment	4

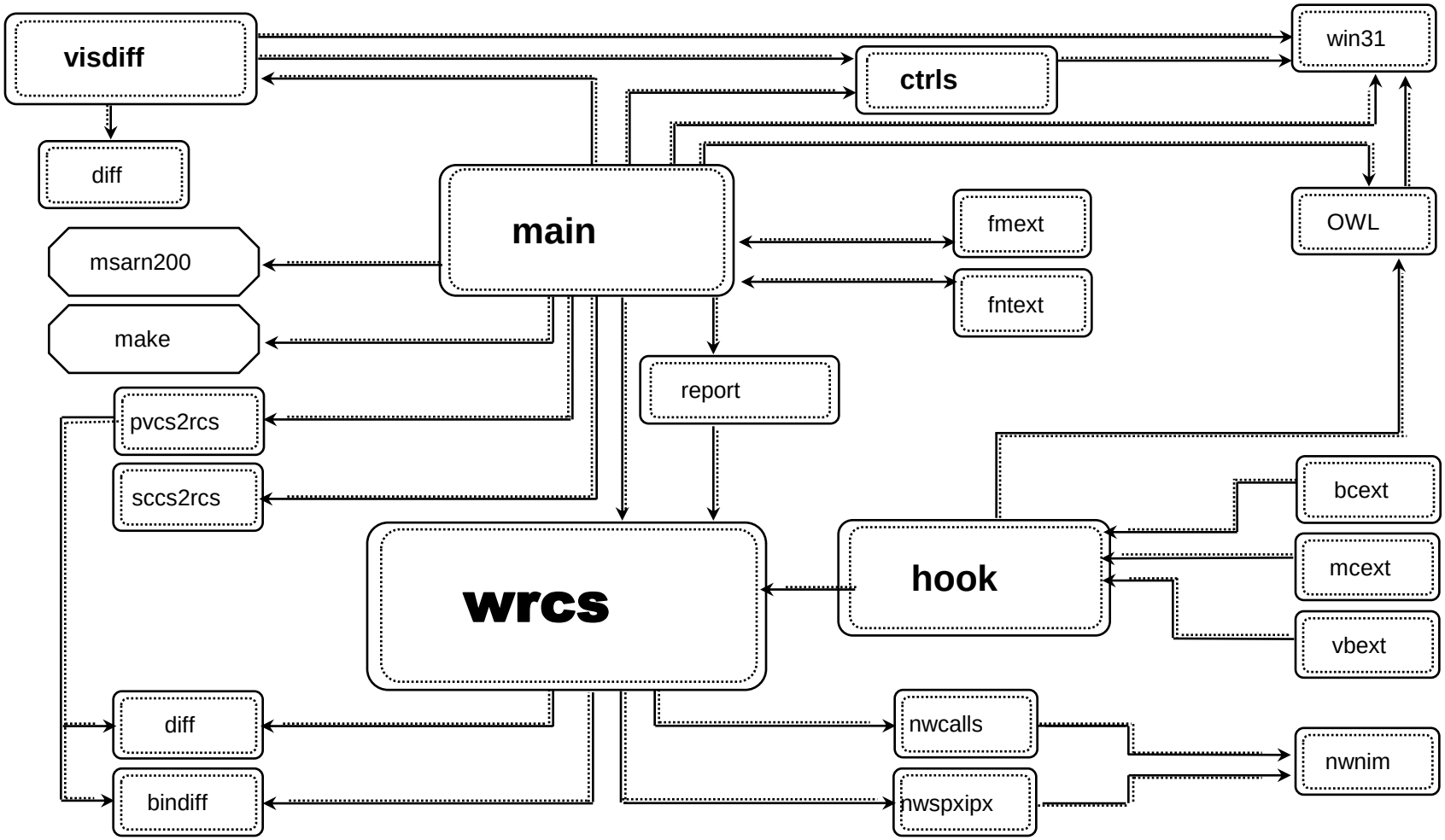
UNIVERSITÄT SALZBURG

Evaluation of interactions between indirect scenarios

Modul	Number of changes
main	4
wrcs	7
diff	1
bindiff	1
pvcs2rcs	1
sccs2rcs	1
nwcalls	1
nwspxipx	1
nwnlm	1
hook	4
report	1
visdiff	3
ctrls	2

UNIVERSITÄT SALZBURG

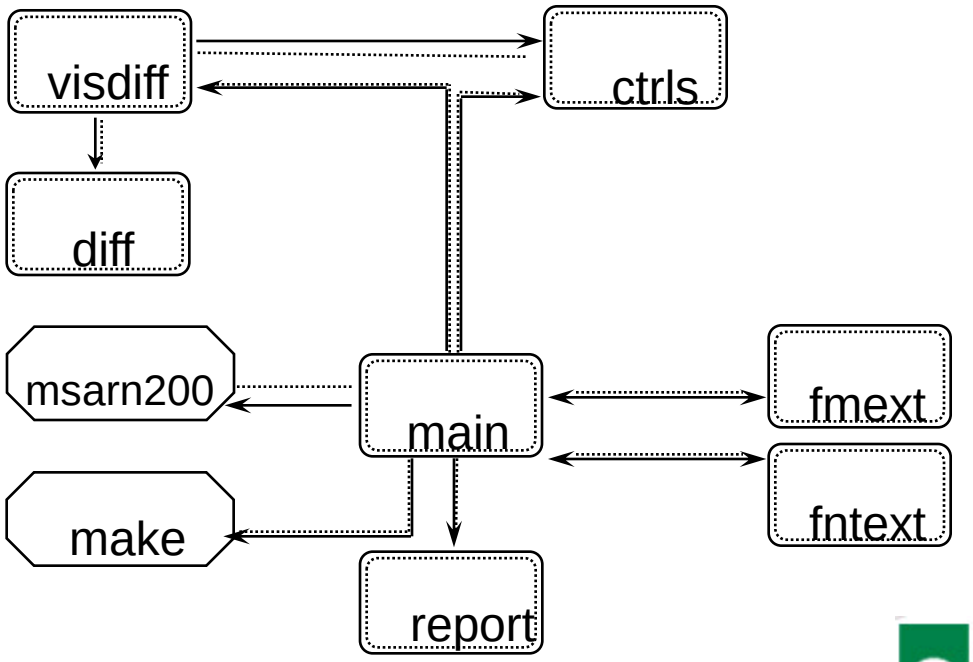
Visualization of scenario interactions



UNIVERSITÄT SALZBURG

Advantages and risks of using SAAM (I)

- The usefulness of the results depends crucially on the correct choice of the granularity of architecture description. For example, the following granularity would be problematic with WRCS:



UNIVERSITÄT SALZBURG

Advantages and risks of using SAAM (II)

- Makes possible rapid and goal-safe evaluation of the quality of a software architecture, in particular regarding changes and extensions.
- No detailed code inspections are necessary.
- An expenditure of few person days (2 to 5 days depending upon system complexity) is necessary.
- Experience and technical knowledge of the stakeholders are conditions for a successful SAAM application.
- Stakeholder participation is the socio-economic component of the SAAM and secures the efficiency of the analysis process.
- The SAAM offers a pragmatic possibility to “measure” coupling and cohesion. Based on these measurements, one can make an evaluation of the balance between coupling and cohesion of a selected modularity.

UNIVERSITÄT SALZBURG

Multidimensional Modularization by *Aspect Oriented Programming (AOP)*

UNIVERSITÄT SALZBURG

AOP as a further development from metaprogramming

- The term AOP was introduced by Gregor Kiczales in the mid-1990s
- The basic idea of AOP is that one should be able to define multiple modularizations for a software system, rather than a single (statically defined) modularization. Thus, AOP is viewed as a means for "multi-dimensional modularization ".
- AOP focuses on the identification, specification and representation of *cross-cutting concerns* and their modularization into separate functional units.
- AOP is an application of meta-programming to improve the modularization of software.

UNIVERSITÄT SALZBURG

AOP example (I)

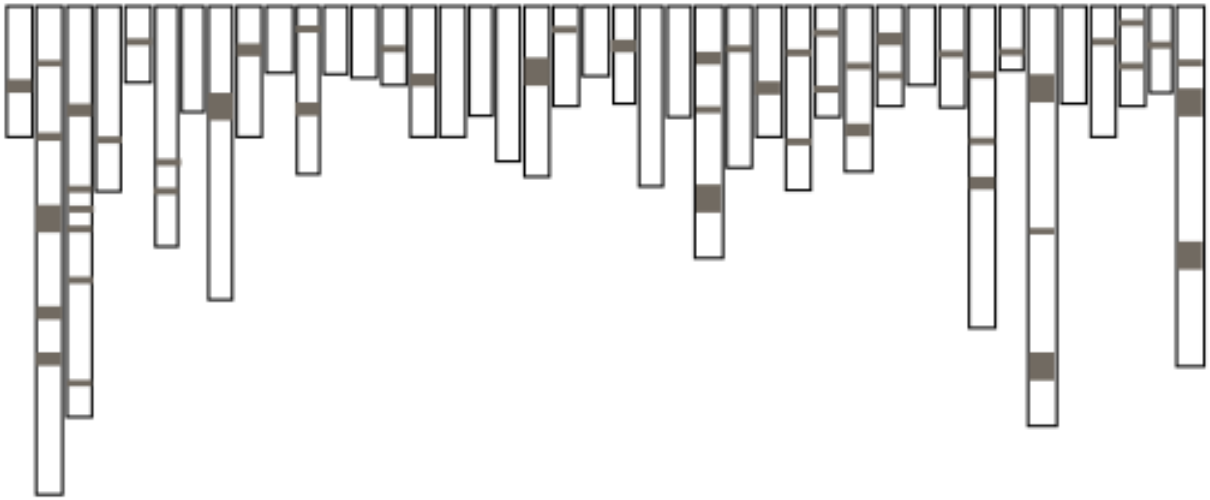
Example: the Apache Tomcat Web-Server

Good modularization of the concern „URL-recognition“

UNIVERSITÄT SALZBURG

AOP example (II)

Example: the Apache Tomcat Web-Server

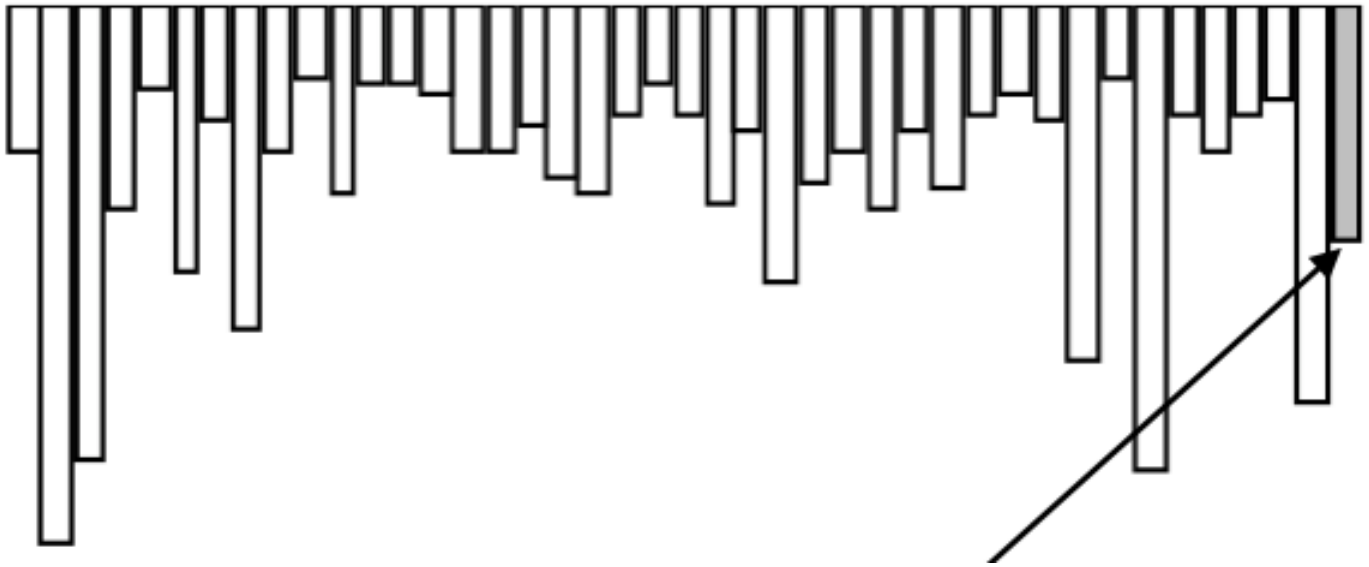


Logging code is scattered across many modules

UNIVERSITÄT SALZBURG

AOP solution: Cross-cutting concerns

Example: the Apache Tomcat Web-Server



A new AOP module „Logging“ with the cross-cutting logging code

UNIVERSITÄT SALZBURG

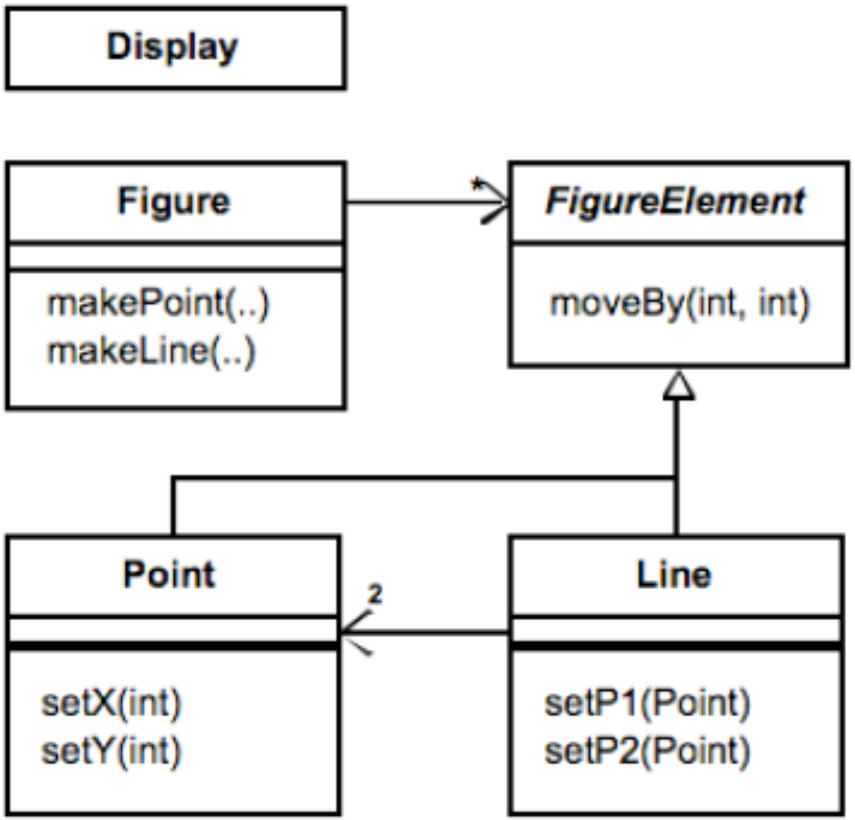
AOP concepts in AspectJ (I)

- Aspect J is the aspect-oriented extension of Java
- It extends the semantics of the Java language with the so-called „Dynamic Join Points“
- It extends the language with four constructs:
  - Point cut
  - Advice
  - Aspect
  - Intra-class declaration

UNIVERSITÄT SALZBURG

AOP concepts in AspectJ (II)
Example: Graphical editor

- Starting point (not AOP)



UNIVERSITÄT SALZBURG

AOP concepts in AspectJ (III)
Example: Graphical editor

- Starting point (not AOP)

```
class Line implements FigureElement {
  private Point p1, p2;
  Point getP1() { return p1; }
  Point getP2() { return p2; }
  void setP1(Point p1) { this.p1 = p1; Display.update(); }
  void setP2(Point p2) { this.p2 = p2; Display.update(); }
  void moveBy(int dx, int dy) { ... Display.update(); }
}
```

UNIVERSITÄT SALZBURG

AOP concepts in AspectJ (IV)
Example: Graphical editor

- Starting point (not AOP)

```
class Point implements FigureElement {
  private int x = 0, y = 0;
  int getX() { return x; }
  int getY() { return y; }
  void setX(int x) { this.x = x; Display.update(); }
  void setY(int y) { this.y = y; Display.update(); }
  void moveBy(int dx, int dy) { ... Display.update(); }
}
```

UNIVERSITÄT SALZBURG

AOP concepts in AspectJ (V)

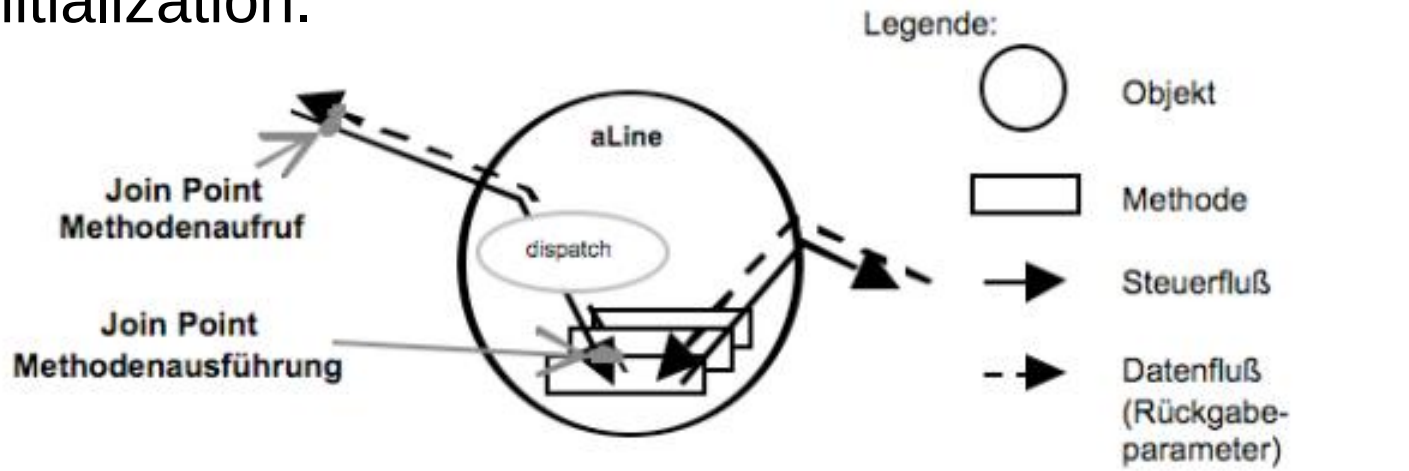
- AOP **join points**: well-defined moments in the execution of a program, like method call, object instantiation, or variable access.

- The AOP developer can thus specify actions to be executed at the join points, for example when a method is called or when a method is executed.

UNIVERSITÄT SALZBURG

AOP concepts in AspectJ (V)
Example: Graphical editor

- The following types of join points can be distinguished:
  - Method and constructor calls and executions
  - Access to instance variables (setter and getter methods)
  - Execution of exception handling, static and dynamic initialization.



UNIVERSITÄT SALZBURG

AOP concepts in AspectJ (VI)
Example: Graphical editor

- A **pointcut** is defined as a logical predicate on a join point. For example, the pointcut

  call (void Line.setP1 (Point))

  is hit when a method with signature

  void Line.setP1(Point)

  is called.

  The AspectJ keyword **call** designates join points for method calls. Pointcuts can be composed by using logical operators (according to Java syntax), such as:

  call (void Line.setP1 (Point)) ||

  call (void Line.setP2 (Point))

UNIVERSITÄT SALZBURG

AOP concepts in AspectJ (VII)
Example: Graphical editor

- A pointcut can be named. In our example, we name the pointcut *move* :

  pointcut move():

  call (void Line.setP1 (Point)) | |
  call (void Line.setP2 (Point))

  A pointcut can also have parameters.

UNIVERSITÄT SALZBURG

AOP concepts in AspectJ (VIII)
Example: Graphical editor

- AspectJ provides more pointcut designators, such as **execution**, **get**, **set**, and **handlers**, which correspond to the other
  join point types:
  - Method execution
  - Access to instance variables
    (setter and getter methods)
  - Execution of exception handling

- Details about these pointcuts are outside the scope of our course.

UNIVERSITÄT SALZBURG

AOP concepts in AspectJ (IX)

- An **Aspect** is defined as a specific class that “cuts across” different classes which contain scattered code that logically belongs to the same module.

- Aspects consist of an association of ordinary variables and methods, pointcut definitions, inter-type declarations, and advice.

UNIVERSITÄT SALZBURG

AOP concepts in AspectJ (X)
Example: Graphical editor

```
aspect  DisplayUpdating {
  pointcut move():
    call(void FigureElement.moveBy(int, int)) ||
    call(void Line.setP1(Point))              ||
    call(void Line.setP2(Point))              ||
    call(void Point.setX(int))                ||
    call(void Point.setY(int));
  after() returning: move() {        // after Advice
    Display.update();
  }
}
```

UNIVERSITÄT SALZBURG

AOP concepts in AspectJ (XI)
Example: Graphical editor

- An **Advice** specifies the code that is executed at a join point matched by a pointcut.
- In the previous example, the *after* keyword states what has to be done at every join point which matches the pointcut *move*: the static method update() of the class Display is executed.
- This behavior is achieved in the conventional, object-oriented implementation, by calls to *update* scattered across methods of classes Point and Line.

UNIVERSITÄT SALZBURG

Other AOP concepts, not detailed in this introductory presentation

- AOP involves many other concepts, which are beyond the scope of this outline:
  - Parameters
  - Pre- and Post-conditions
  - Wildcards
  - Reflection
  - Abstract Aspects
  - AOP embedding in programming environments

UNIVERSITÄT SALZBURG

Critical view of AOP

- A disadvantage of AOP is that locality is lost in further developments. For example, when a new FigureElement is defined (e.g., a Circle), the aspect DisplayUpdating must also be modified, in addition to the „global class“.

- The more cuts are planned, the more difficult it is to understand which predicates are true and when. It becomes harder to see “the big picture”.

UNIVERSITÄT SALZBURG

Summary of important modularization principles

UNIVERSITÄT SALZBURG

Important modularization principles

- The most important property of a module is the means for abstractions that it makes available. Unimportant (implementation) details should be hidden behind an interface.
- The definition of a software architecture through the modularization of a software system requires a good knowledge of the application field. The software components should match the entities of the application field.
- The modules should achieve a balance between coupling and cohesion.

UNIVERSITÄT SALZBURG

Regarding architectures...

- The modularization of a software system is a difficult, partly artistic task. Similarly to Building Architecture, there is no method that always achieves a good result.
- Christopher Alexander described patterns for building good architectures and coined the term "quality without a name“: A good architecture has a "quality without a name" that cannot be described explicitly.
- To become a good architect, you should study many good architectures, which have a "quality without a name”. Unfortunately, there are only a few such architectures available.
- Look at academic and open source projects!

UNIVERSITÄT SALZBURG