OO concepts
UML representation

- Objects, Classes, Messages/Methods
- Inheritance, Polymorphism, Dynamic Binding
- **Abstract Classes, Abstract Coupling**

UNIVERSITÄT SALZBURG

Abstract Classes and Abstract Coupling

UNIVERSITÄT SALZBURG

Why abstract classes?

- OO languages are used in many software projects in the same manner as module-oriented languages (Modula-2, Ada) for Lego-style building from separate parts:
  - Classes are a language construct for implementing modules/abstract data types
  - Such software modules can be adapted to new projects by sub-classing
- In order to achieve **reusable software architectures,** it is essential to employ a skillful combination of subclassing (and thus polymorphism) and dynamic binding in the form of abstract classes.

UNIVERSITÄT SALZBURG

Abstract classes (I)

- Provide default (general) behavior
  - Only few methods are implemented
- Require subclasses to provide more specific behavior
  - Some method names and their parameters are fixed, but their implementation must be provided by subclasses
- Represent a standardization of the class interface for all subclasses

UNIVERSITÄT SALZBURG

Abstract classes (II)

- The classes resulted from grouping common characteristics **do not reflect usually objects of the real world**, but an abstractization of them. Therefore one calls these classes abstract classes.

- A further reason for this naming is that **it does not makes sense to generate instances** from such classes. Abstract classes contain “dummy” implementations or no implementations (→ abstract methods) for some methods.

UNIVERSITÄT SALZBURG

Abstract Coupling (I)

- Other classes can be implemented based on abstract classes. The **coupling** between a class (e.g. class B) and an abstract class (e.g. class A) can take place in several ways:
  - B has an instance variable of the static type A
  - One or more methods of B have a parameter of the static type A
  - B accesses a global variable of the static type A

UNIVERSITÄT SALZBURG

Abstract coupling (II)

- The classes coupled with an abstract class can work with objects of arbitrary subclasses of the abstract class **without change**. This is due to polymorphism and dynamic binding.
- The behavior of these components can thus be changed not by changing the components themselves, but by technically clean modifications of the behavior of abstract classes, which is done in their subclasses.

Abstract classes + abstract coupling = the basis for OO Frameworks (semifinished designs).

UNIVERSITÄT SALZBURG

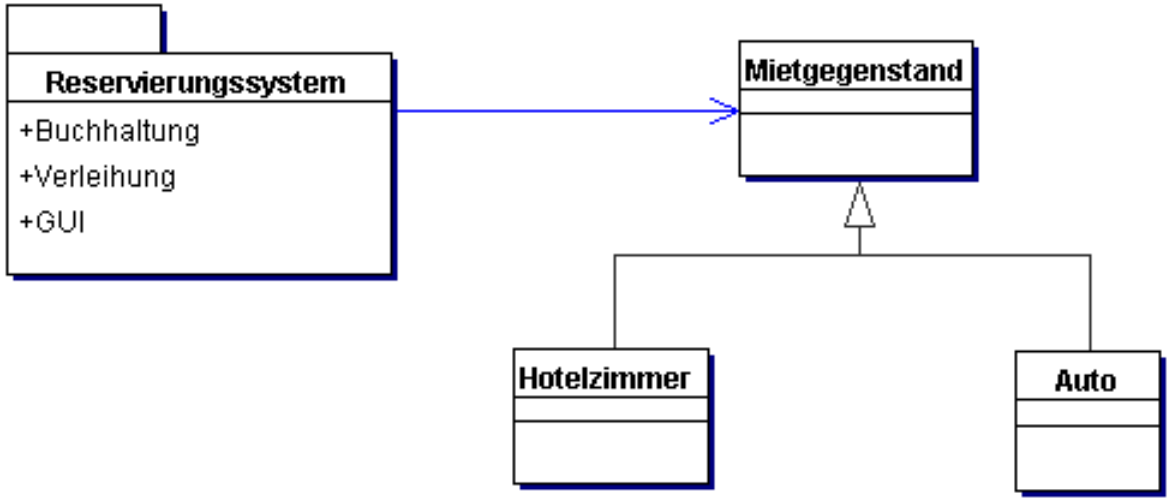
Abstract coupling (III)

The **main issue** is to find **good abstractions** so that other software components can be realized by building on the abstractions.

Abstract classes evolve typically only in interaction with the classes coupled with them.

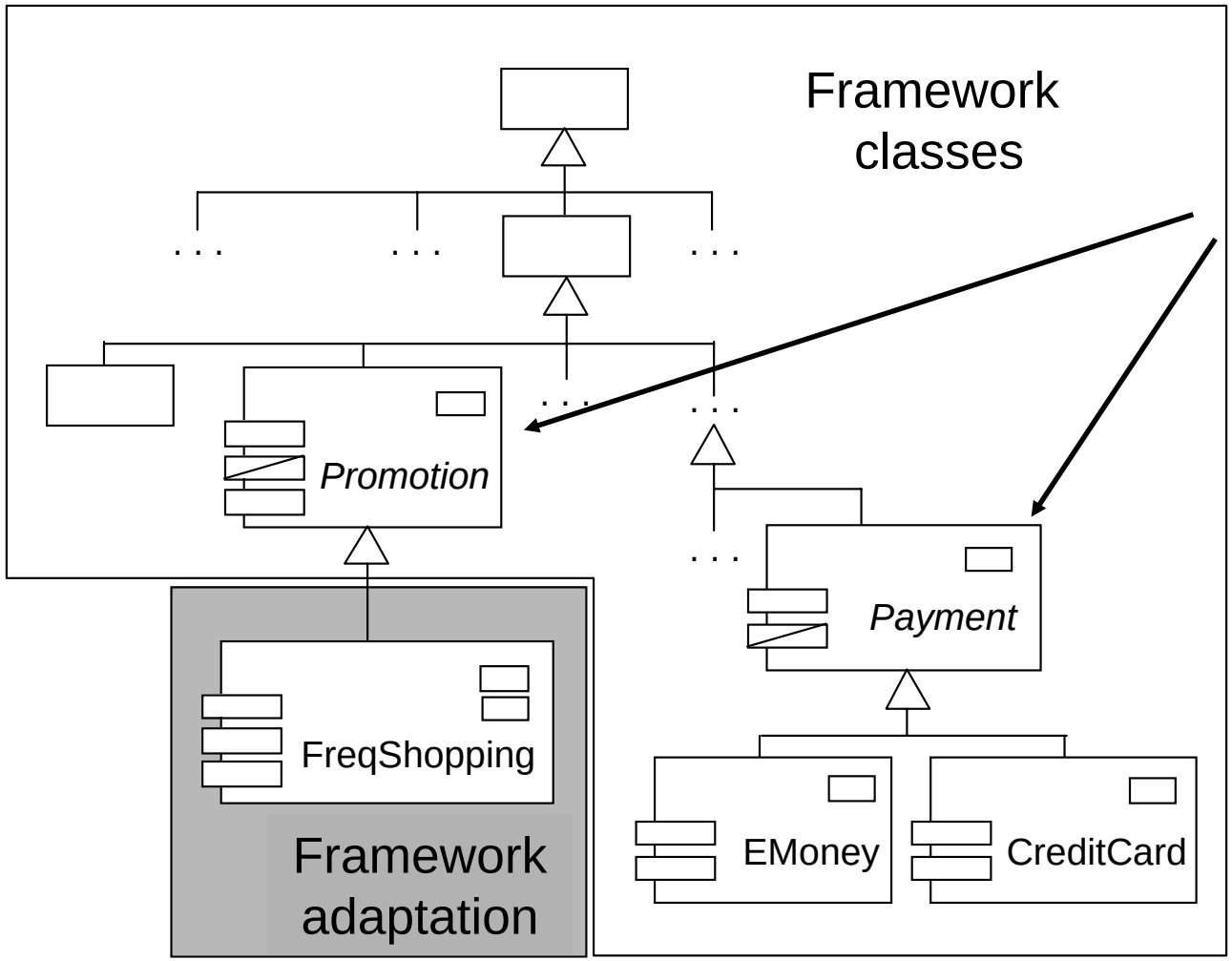
UNIVERSITÄT SALZBURG

Example: reservation system



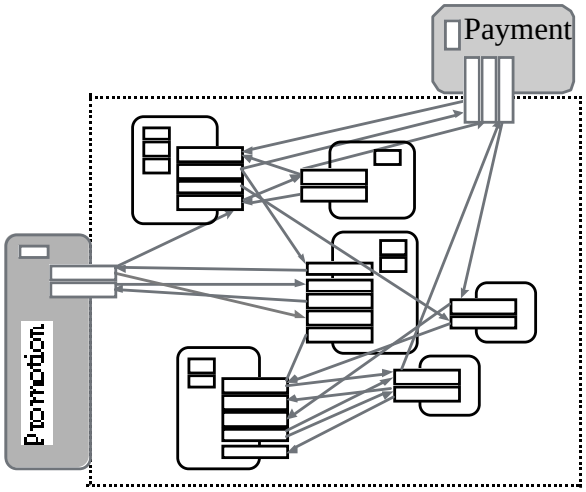
UNIVERSITÄT SALZBURG

Frameworks – static view

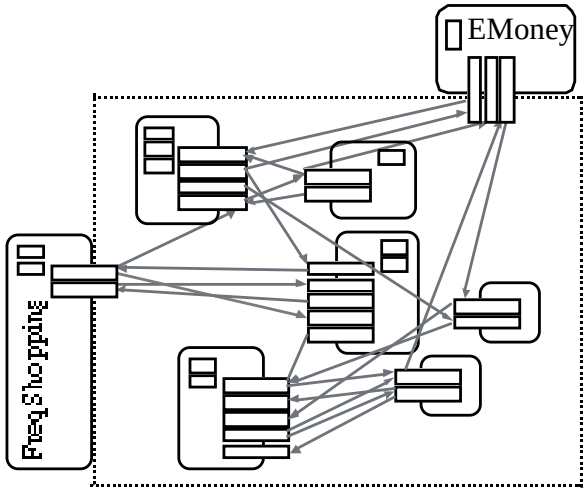


abstract classes (here each with one abstract method)

Black-Box versus White-Box Framework types



vor der Adaptierung



nach der Adaptierung

UNIVERSITÄT SALZBURG

Web shop

UNIVERSITÄT SALZBURG

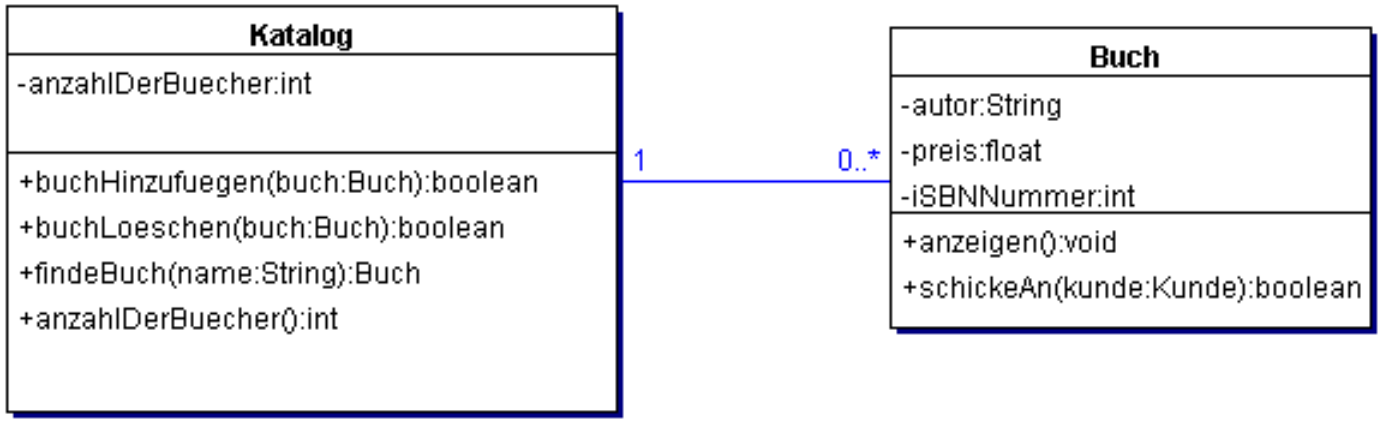
Case Study Web Shop (I)

- Design a Web shop from which one can buy books over Internet.
- A component should be „the catalog“, which administers the books

UNIVERSITÄT SALZBURG

Case Study Web Shop (II)

- First design:



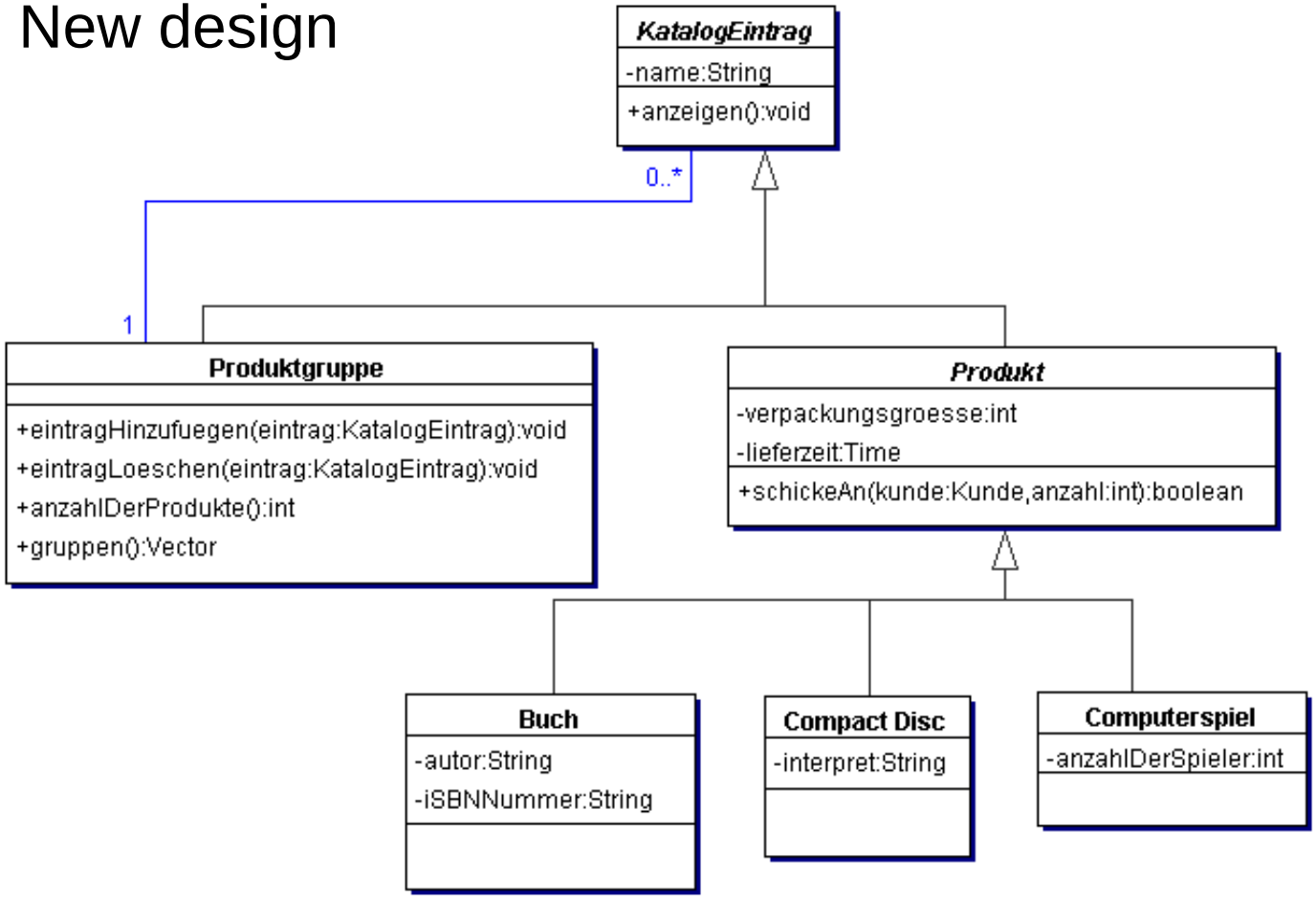
Case Study Web Shop (III)

- Problem: Now also CDs and computer games are to be ordered. The catalog must be extended accordingly.

UNIVERSITÄT SALZBURG

Case Study Web Shop (IV)

- New design



UNIVERSITÄT SALZBURG

Collaboration and Sequence Diagrams

UNIVERSITÄT SALZBURG

Collaboration Diagram (I)

- In a Collaboration diagram there are only simple relationships between objects

  (————————)

- Optionally, the message flow between objects can be represented. (However, sequence diagrams are more suitable in this respect.)

  message, ...

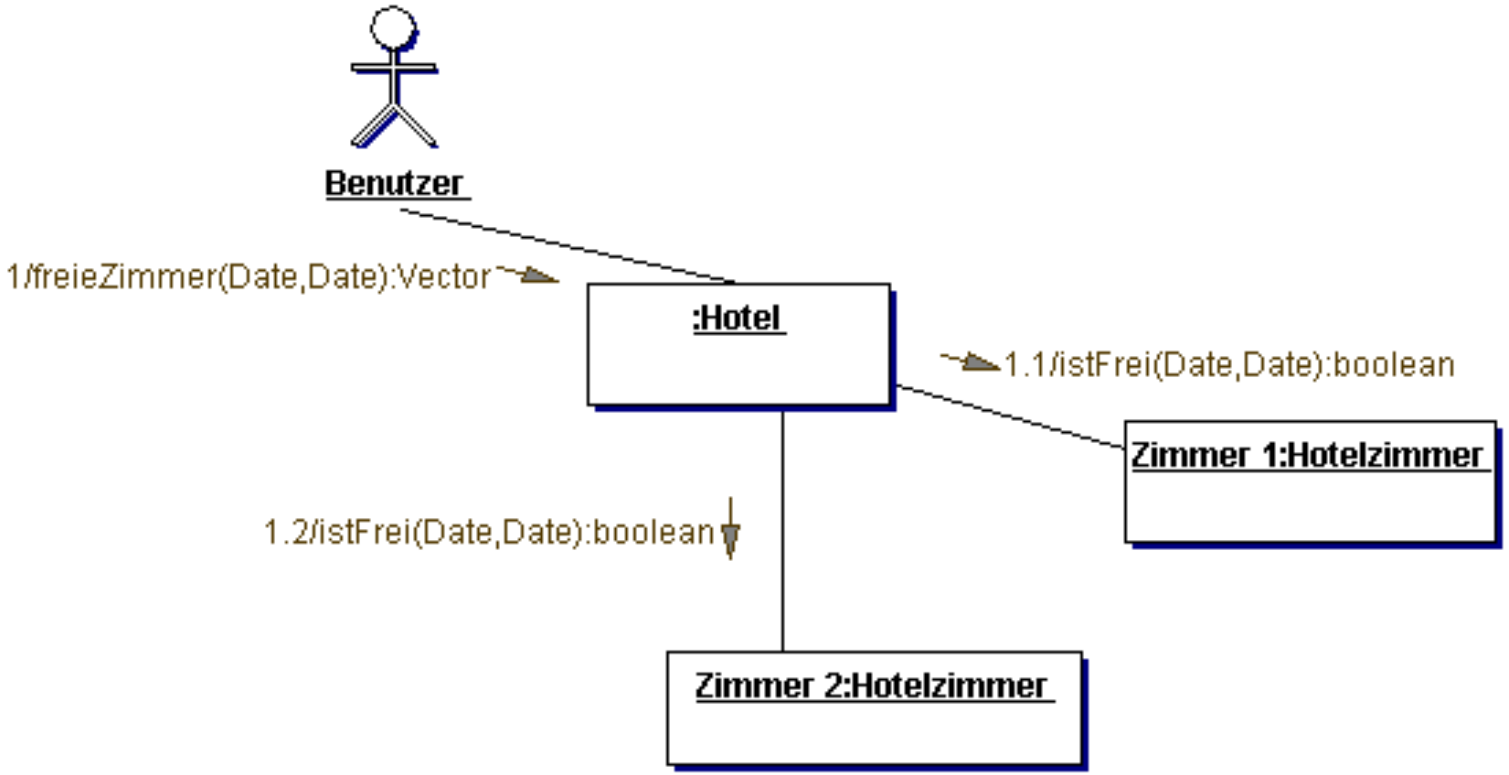
Message is: **[no:] method()**

- **method()** is as given in the class diagram
- **no** is an optional number, which defines the order of the method calls.

UNIVERSITÄT SALZBURG

Collaboration Diagram (II)

Example:



UNIVERSITÄT SALZBURG

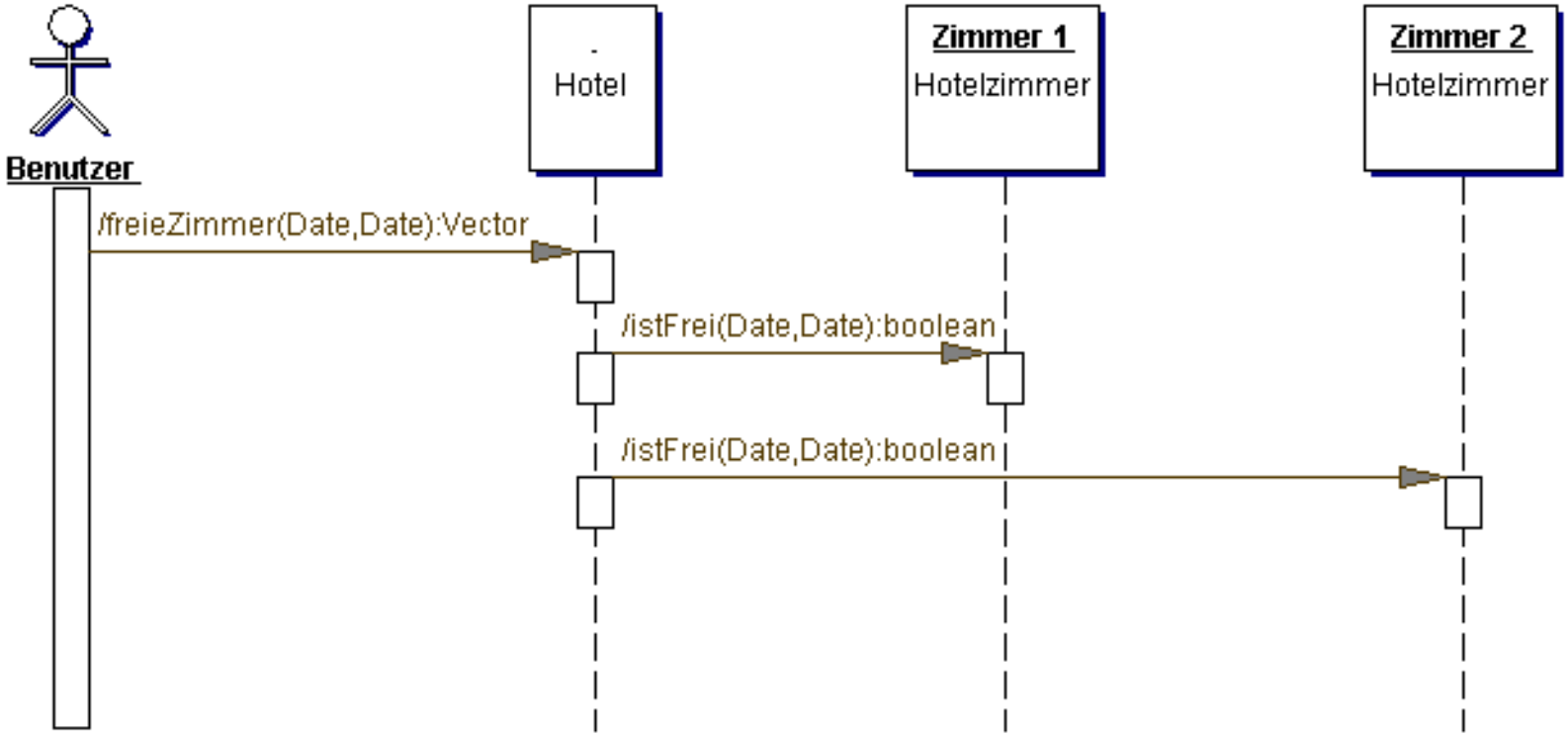
Sequence Diagram (I)

- A sequence diagram essentially expresses the same semantics as a collaboration diagram, but it is usually easier to read
- Collaboration diagrams offer the advantage that additional information can be represented (e.g. relations between objects)
- Collaboration diagrams can be transformed automatically into sequence diagrams.

UNIVERSITÄT SALZBURG

Sequence Diagram (II)

Example:



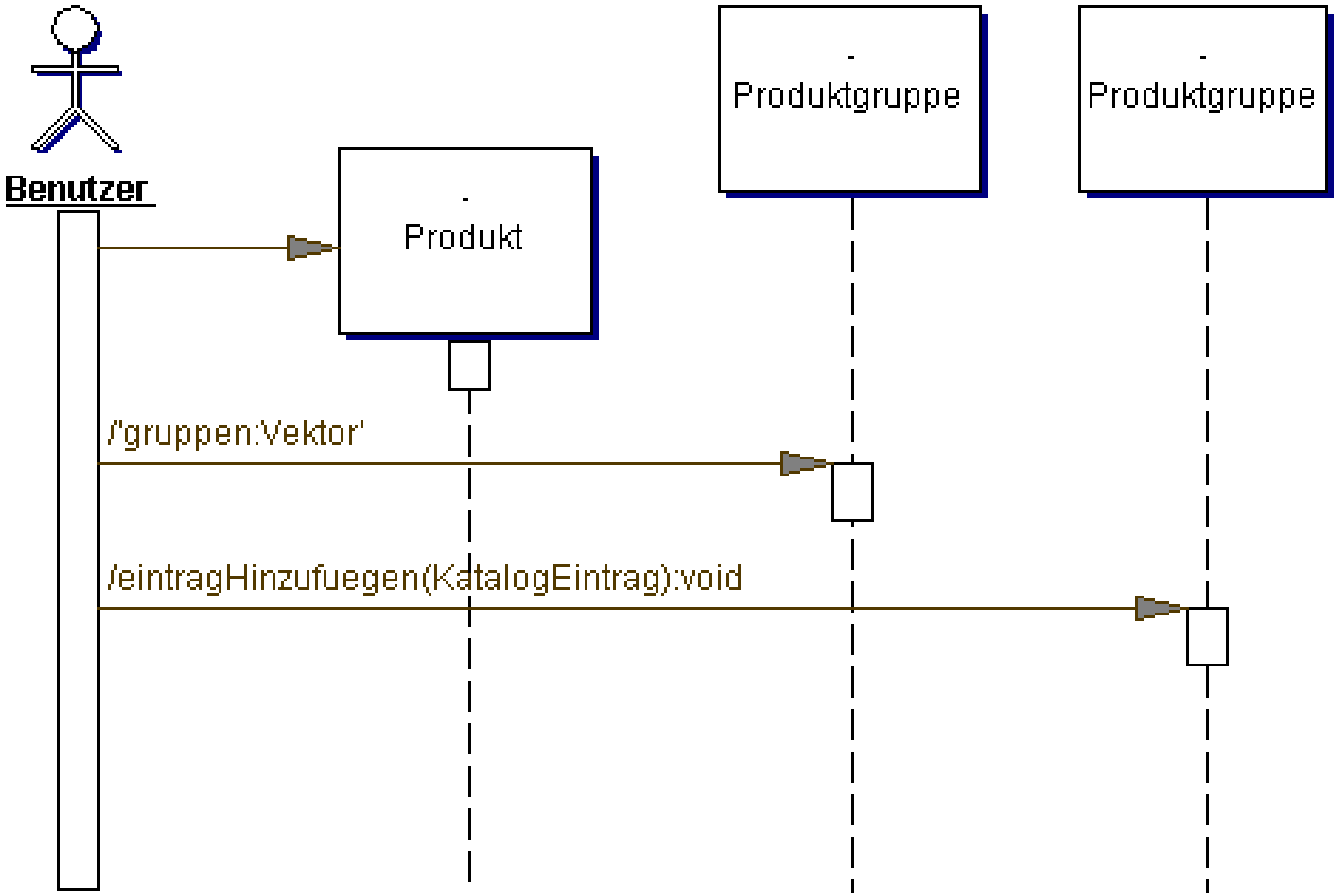
UNIVERSITÄT SALZBURG

Case Study: Web Shop(I)

- Catalog dynamics:
  - How are products inserted into the catalog?
  - How does the sequence diagram look?
  - How does the collaboration diagram look?

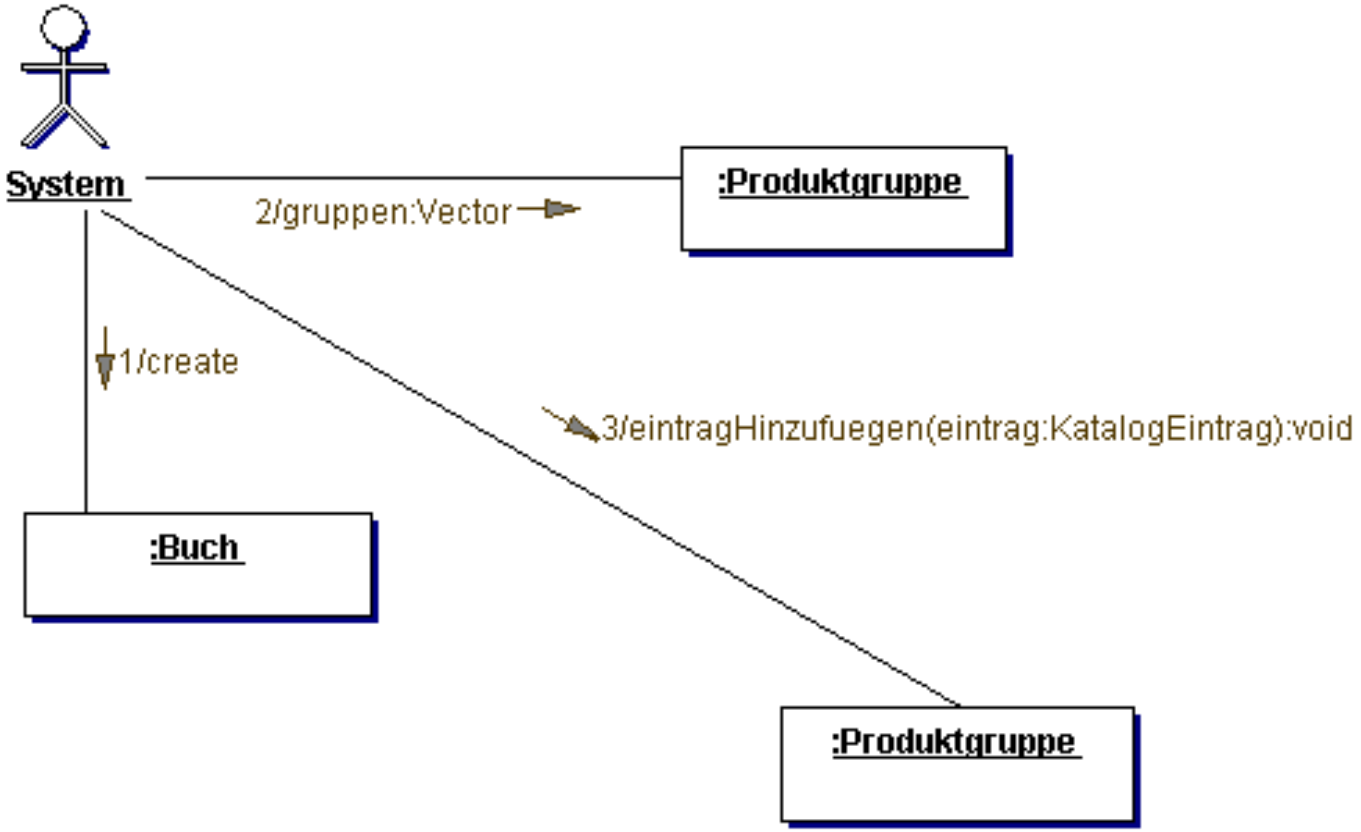
UNIVERSITÄT SALZBURG

Case Study: Web Shop(II)



UNIVERSITÄT SALZBURG

Case Study: Web Shop(III)



UNIVERSITÄT SALZBURG