2 Development process for multimedia projects

2.1 Classical models of the software development process

2.2 Special aspects of multimedia development projects

2.3 Example: The SMART process

2.4 Agile Development and Extreme Programming for multimedia projects

2.5 Modeling of multimedia applications

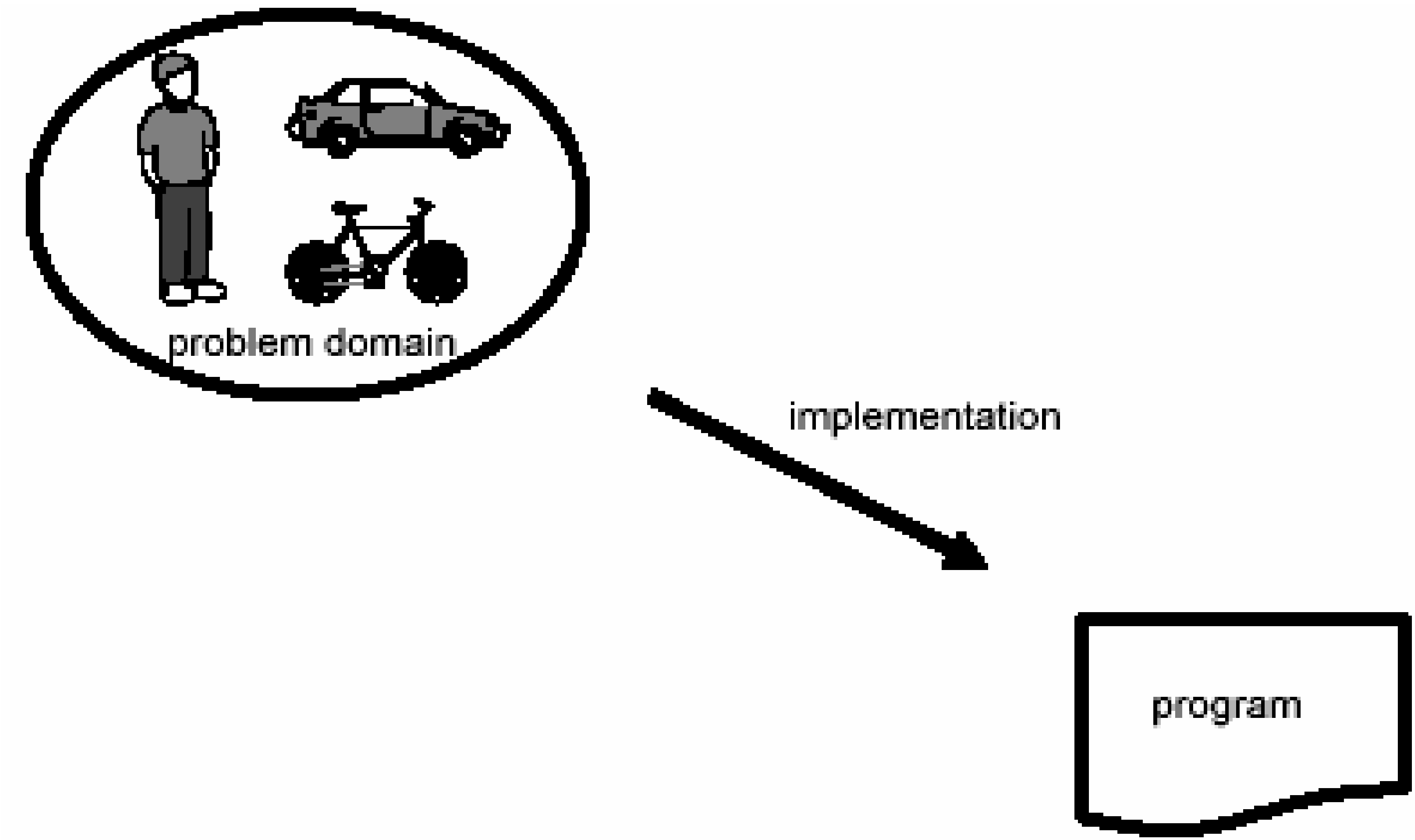
Introduction

Revision: Unified Modeling Language

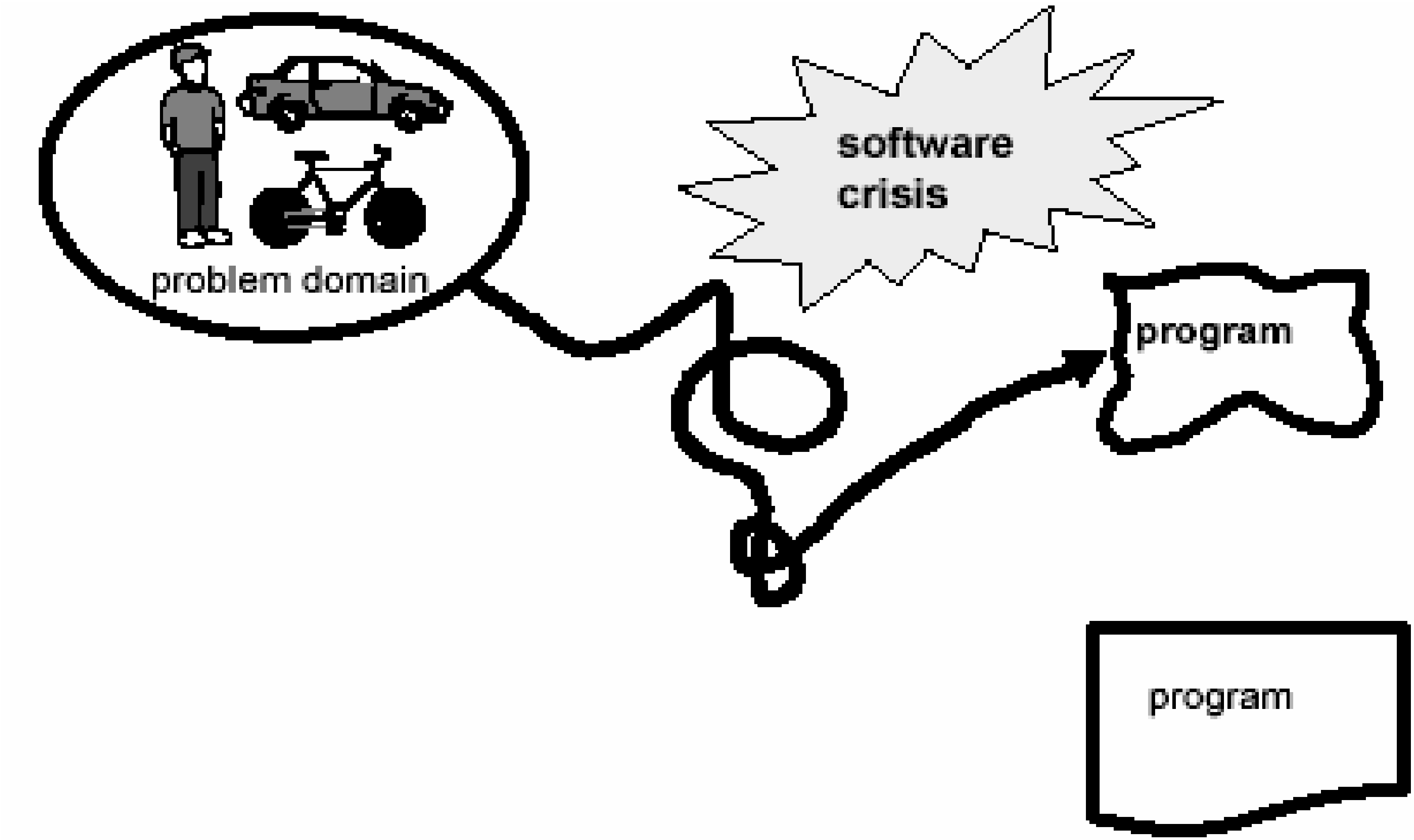
Model-driven architecture

Modeling of multimedia applications

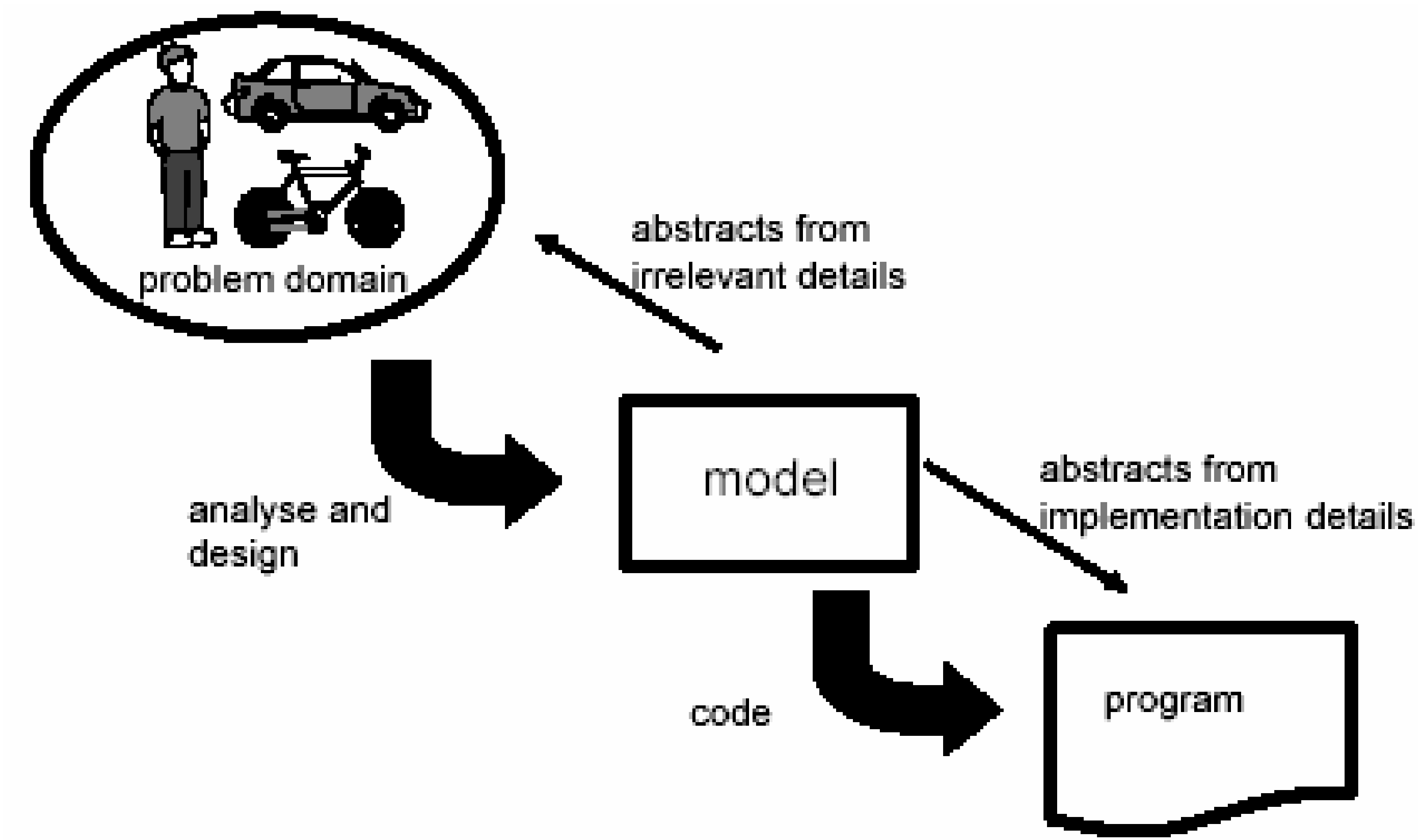
Software Development



Software Development in Practice



Bridging the Gap



Goals of Modeling Approaches

- Communication
  - Within the development team
  - Between different developer groups, e.g. programmer and user interface designer
  - With the customer
  - Model can act as a kind of contract
- Improving the implementation
  - Structuring the code to achieve
    - » better separation into different parts
    - » better maintanability
    - » reusability
  - Omitting errors before implementation
- Notation of models as visual diagrams:
  - More compact and fomal than natural language
  - Easier to handle and to understand than mathematics/logic
  - Often referred to as *semi-formal*

2 Development process for multimedia projects

2.1 Classical models of the software development process

2.2 Special aspects of multimedia development projects

2.3 Example: The SMART process

2.4 Agile Development and Extreme Programming for multimedia projects

2.5 Modeling of multimedia applications

Introduction

Revision: Unified Modeling Language (in german language)

Model-driven architecture

Modeling of multimedia applications

Unified Modeling Language (UML)

OOD
Booch

OOSE
Jacobson

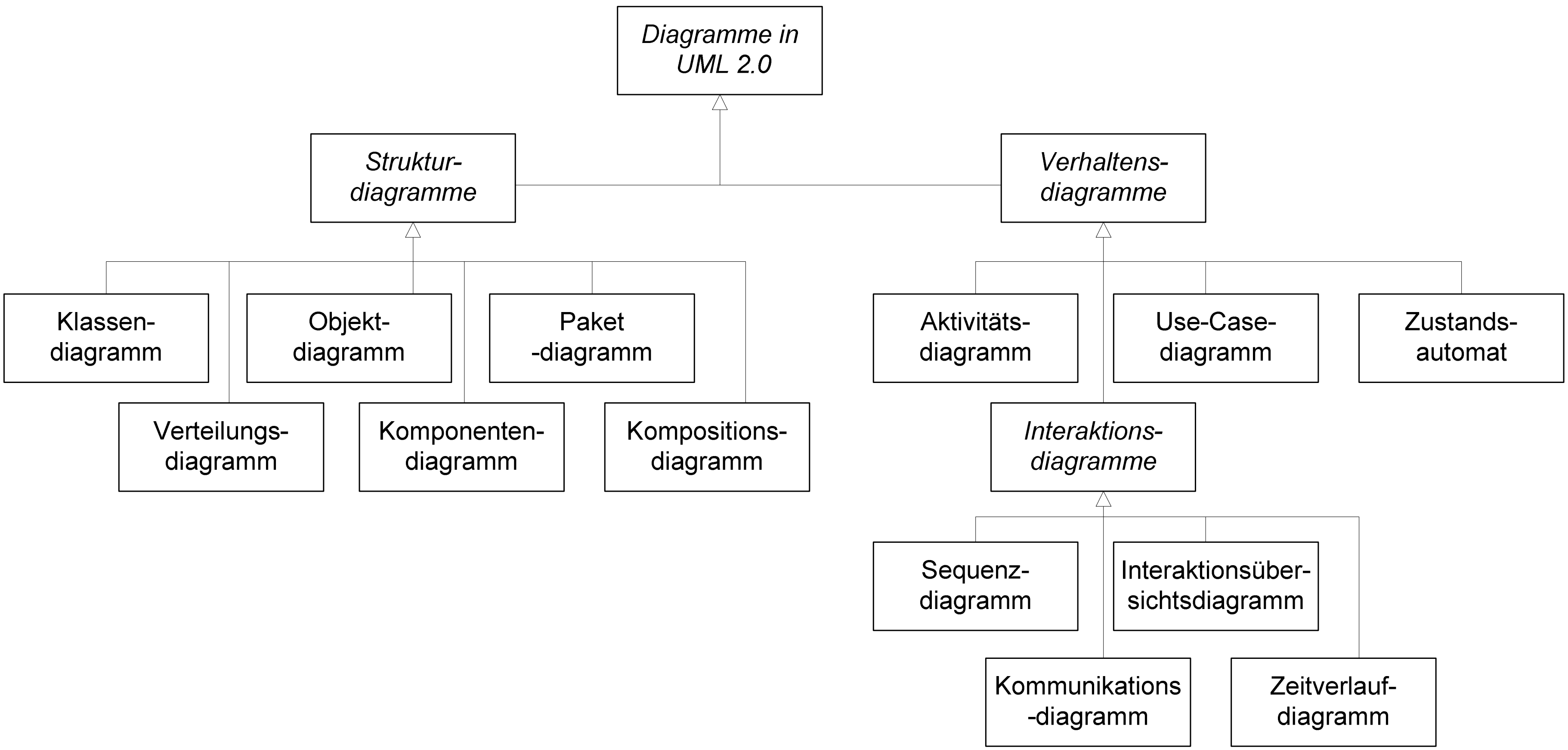
OMT
Rumbaugh et al.

1995: Booch / Rumbaugh / Jacobson

1997:	**UML 1.1**
1999:	**UML 1.3**
2001:	**UML 1.4**
2003:	**UML 2.0**

- UML ist eine Notation der 2. Generation für objektorientierte Modellierung; ursprünglich entwickelt von der Firma Rational
- UML ist Industriestandard der OMG (Object Management Group)

UML 2.0: 13 Diagrammtypen

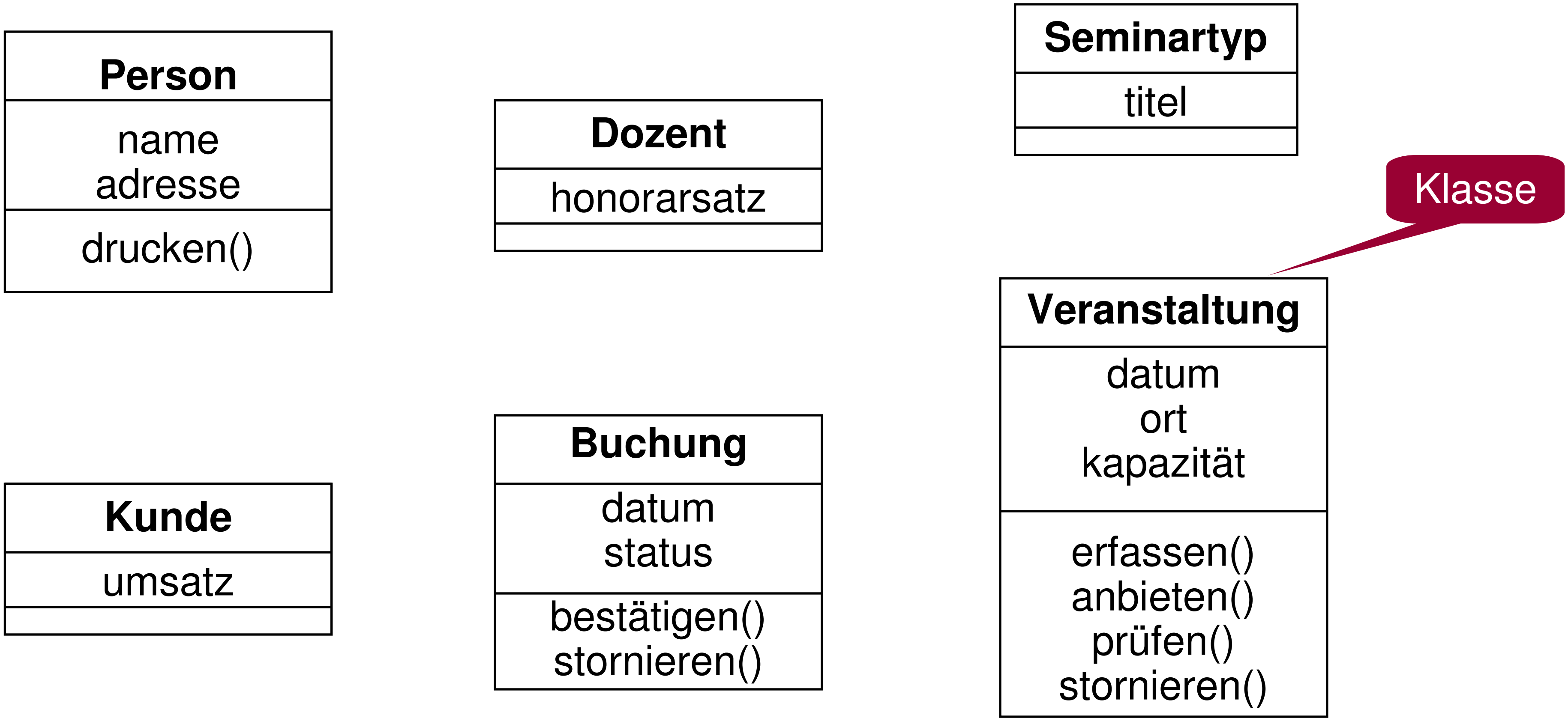


Beispiel

System zur Kunden- und Seminarverwaltung

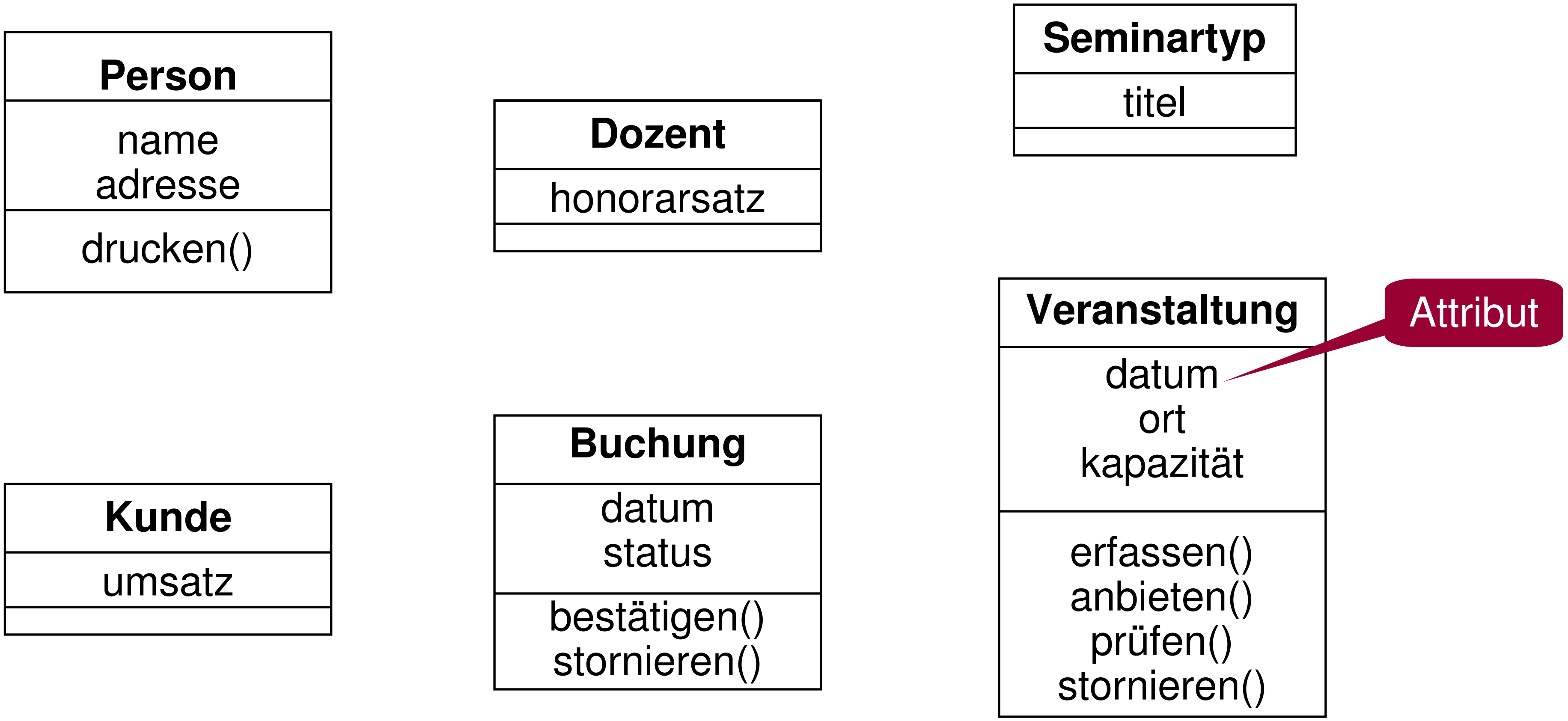
- Zielgruppe: Mitarbeiter einer Schulungsfirma
- Datenhaltung:
  - Kundendaten
  - Dozentendaten
  - Veranstaltungsdaten
  - Buchungsdaten
- Funktionen:
  - Veranstaltungen anbieten, stornieren etc.
  - Kundenbuchungen vornehmen, reservieren etc.
  - ...

Klassendiagramm: Klassen



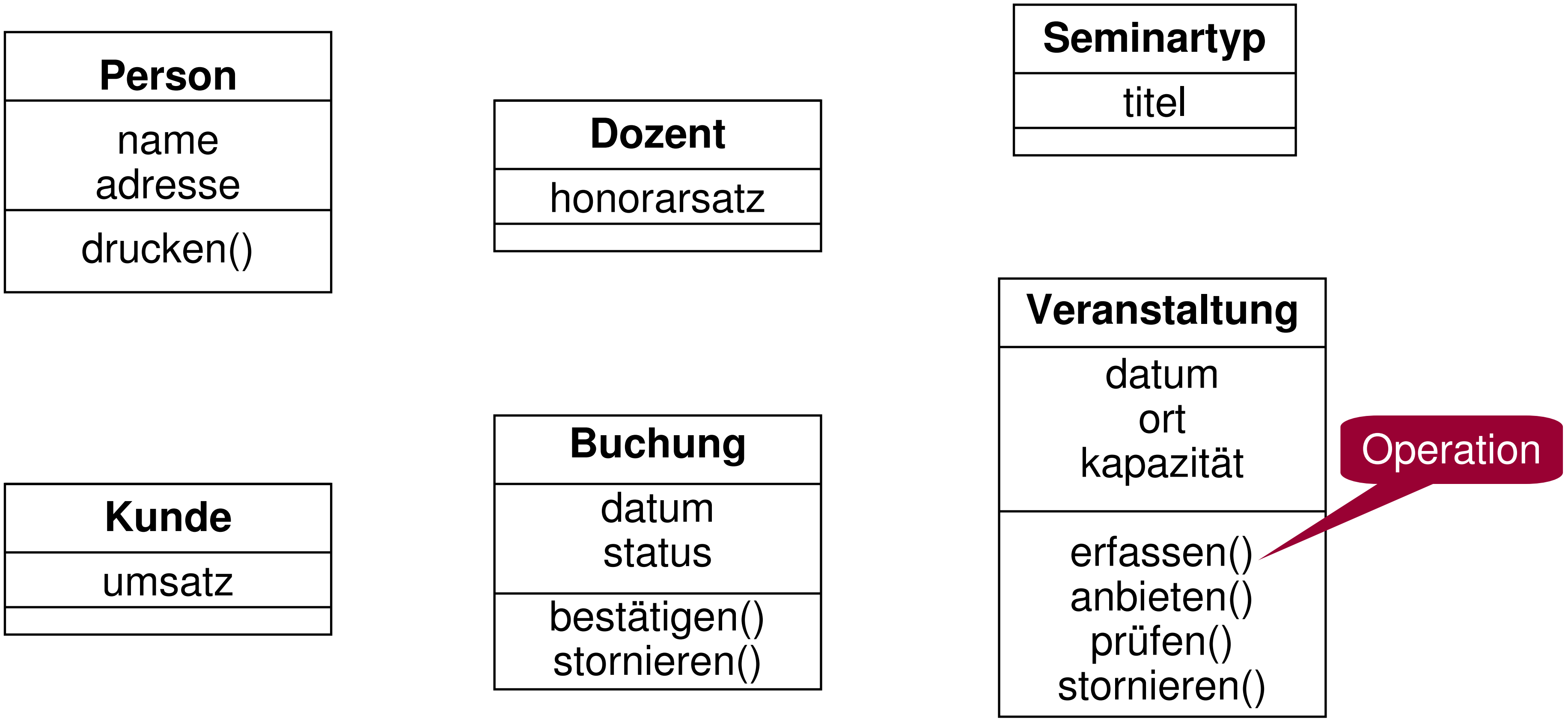
Definition: Eine *Klasse* ist eine Beschreibung gleichartiger Objekte. Ein Objekt wird erzeugt und behält eine unveränderliche *Objektidentität* bis zu seiner Löschung. Jedes Objekt gehört zu (ist *Instanz* von) genau einer Klasse.

Klassendiagramm: Klassen



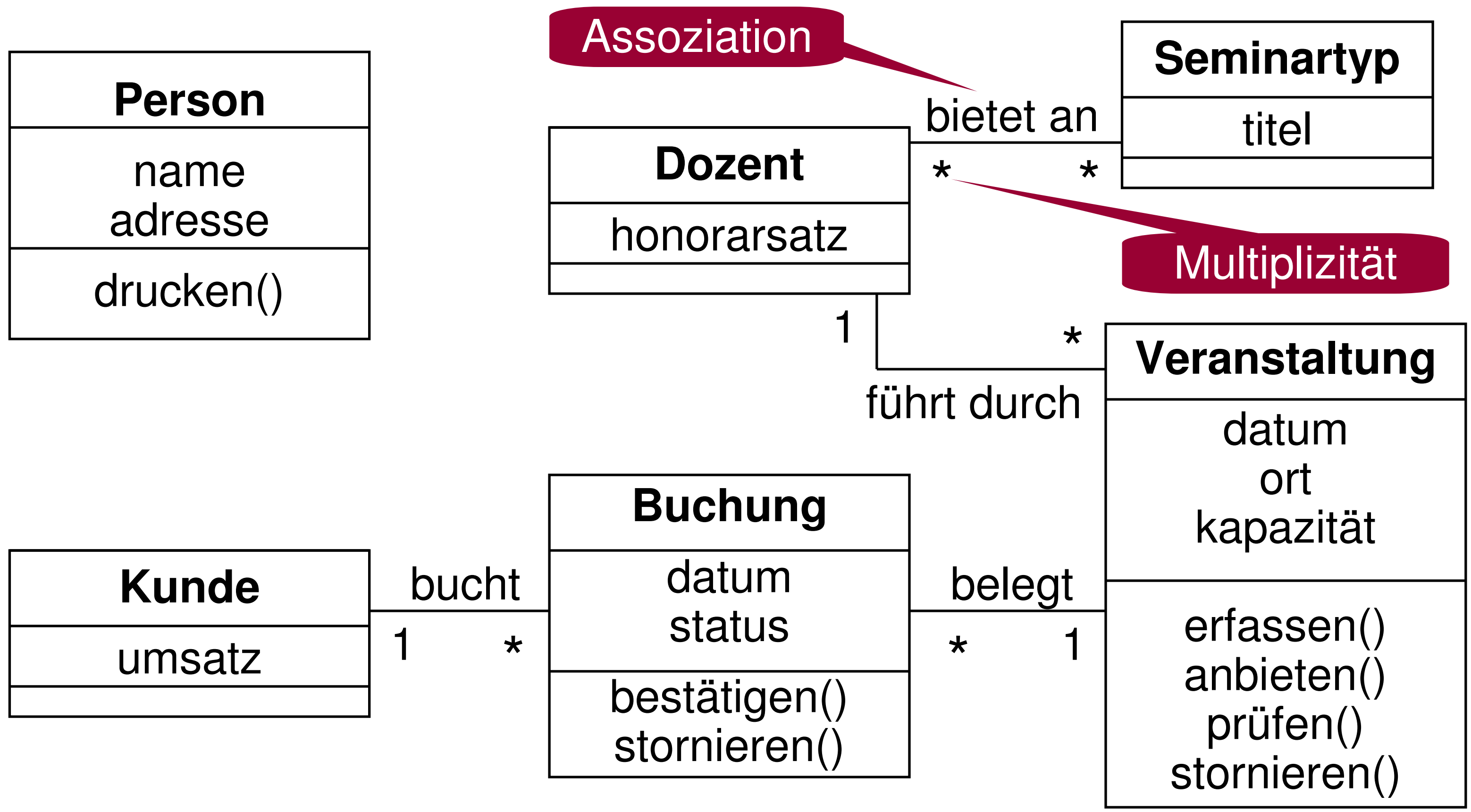
Definition: Ein *Attribut A* einer Klasse *K* beschreibt ein Datenelement, das in jedem Objekt der Klasse vorhanden ist. Jedes Objekt der Klasse *K* trägt für jedes Attribut *A* von *K* einen individuellen und veränderbaren Attributwert.

Klassendiagramm: Klassen



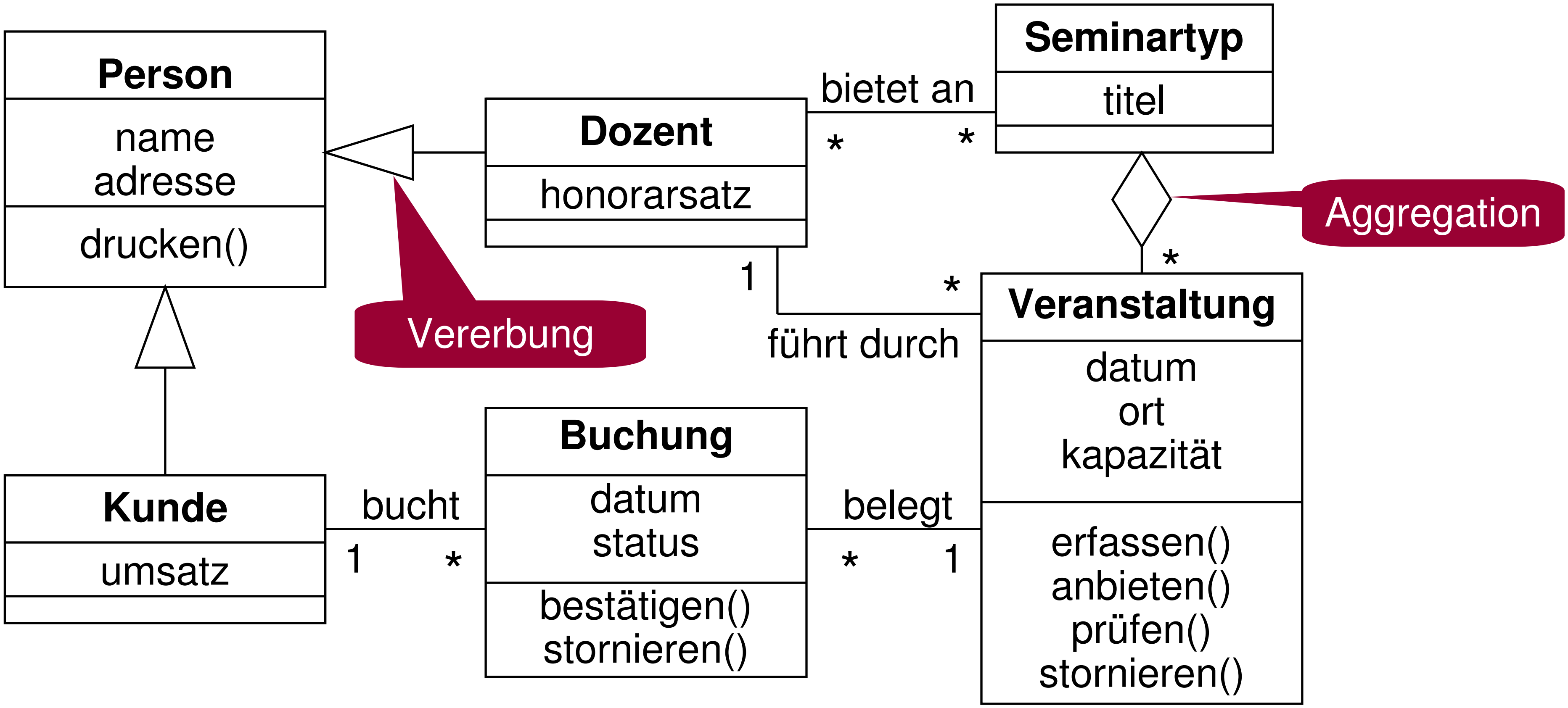
Definition: Eine *Operation* einer Klasse *K* ist die Beschreibung einer Aufgabe, die jede Instanz der Klasse *K* ausführen kann.
In der Beschreibung der Klasse wird der Name der Operation angegeben..

Klassendiagramm: Assoziationen



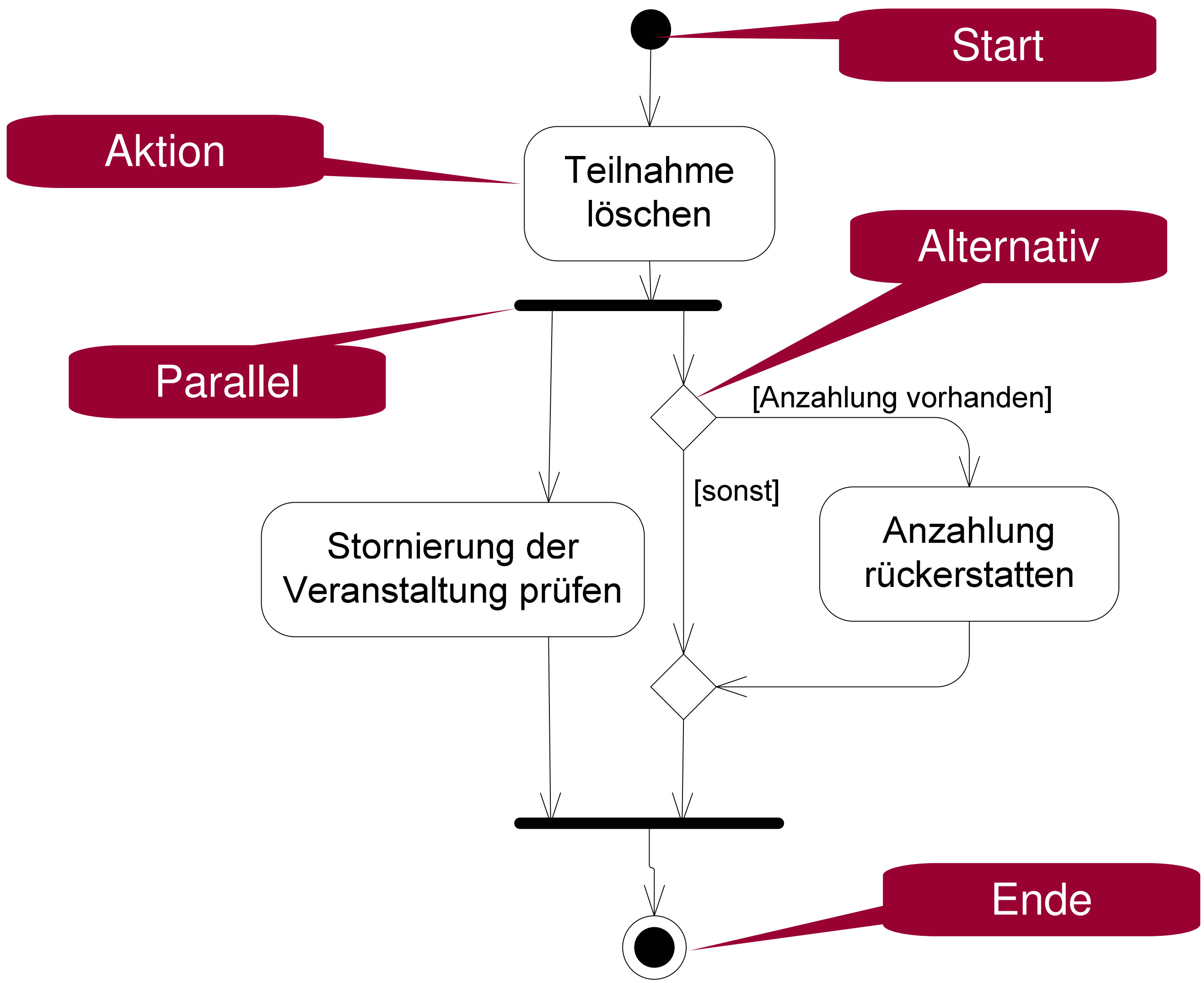
Definition: Die *Multiplizität* einer Klasse *K1* in einer Assoziation mit einer Klasse *K2* begrenzt die Anzahl der Objekte der Klasse *K2*, mit denen ein Objekt von *K1* in der Assoziation stehen darf.

Klassendiagramm: Aggregation, Vererbung

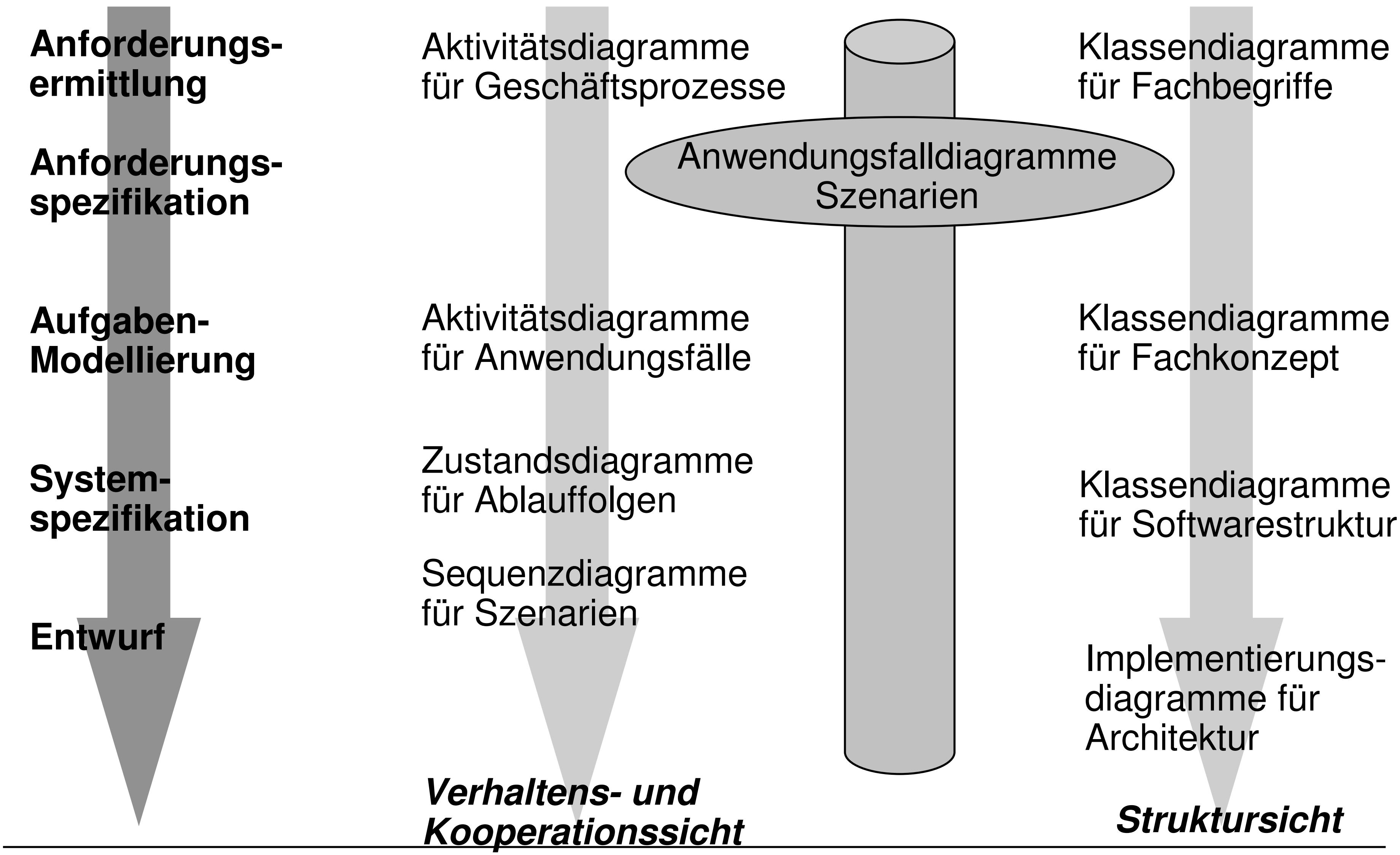


Definition: Eine *Vererbungsbeziehung* von einer Klasse *K1* zu einer Klasse *K2* ist eine Beschreibung der Tatsache, daß alle Objekte der Klasse *K2* zusätzlich zu den in der Klasse *K2* beschriebenen Eigenschaften auch alle Eigenschaften der Klasse *K1* haben.

Aktivitätsdiagramm



UML-Diagrammtypen im Projektverlauf



Klassendiagramme und Softwarestruktur

Person
name: String adresse: String
drucken()

Java:

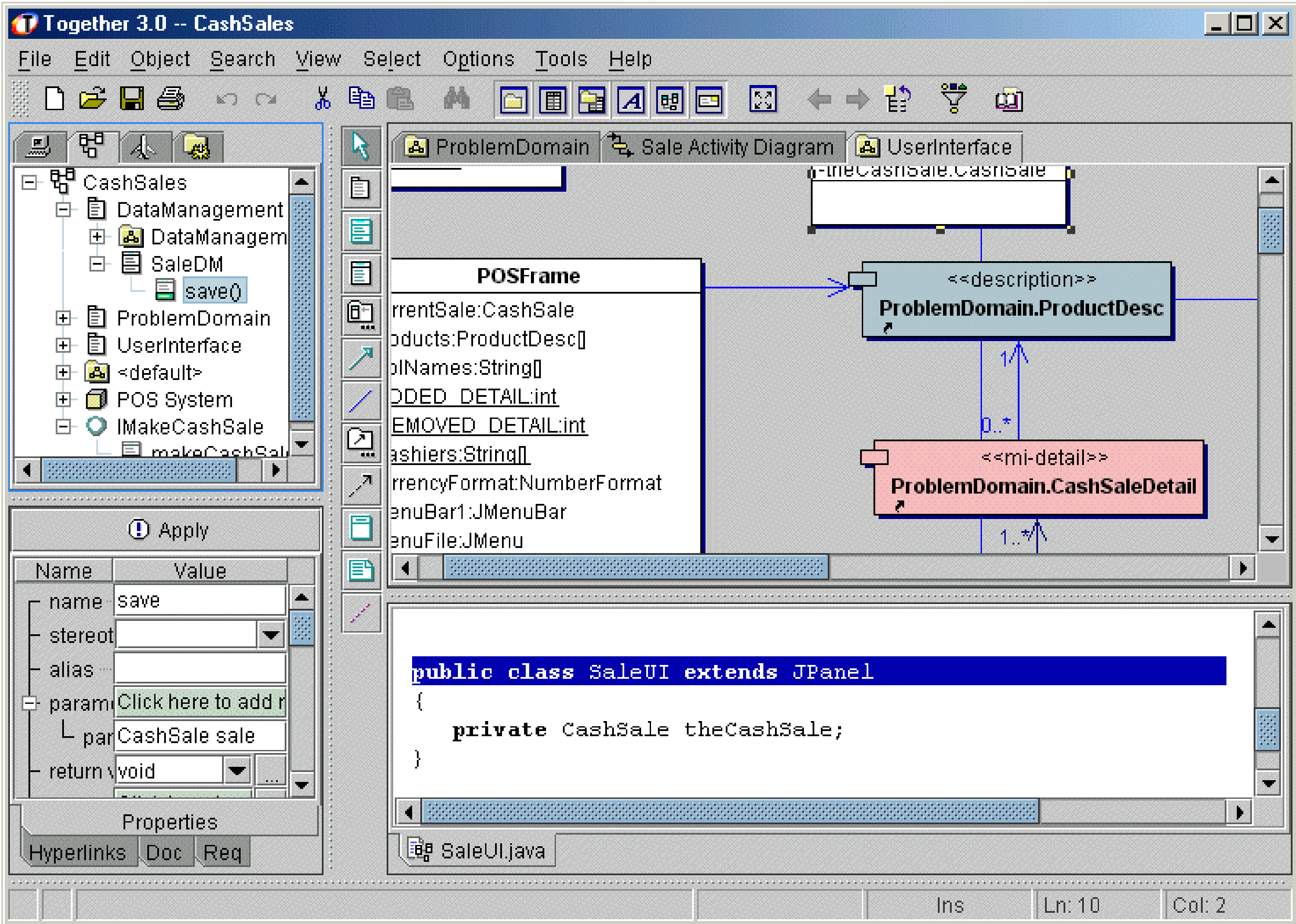
```
class Person {

  String name;
  String adresse;

  void drucken() {
   ...
  }
}
```

Fortgeschrittene Abbildungen: z.B. auf Komponenten (Enterprise Java Beans)

Werkzeugunterstützung



2 Development process for multimedia projects

2.1 Classical models of the software development process

2.2 Special aspects of multimedia development projects

2.3 Example: The SMART process

2.4 Agile Development and Extreme Programming for multimedia projects

2.5 Modeling of multimedia applications

Introduction

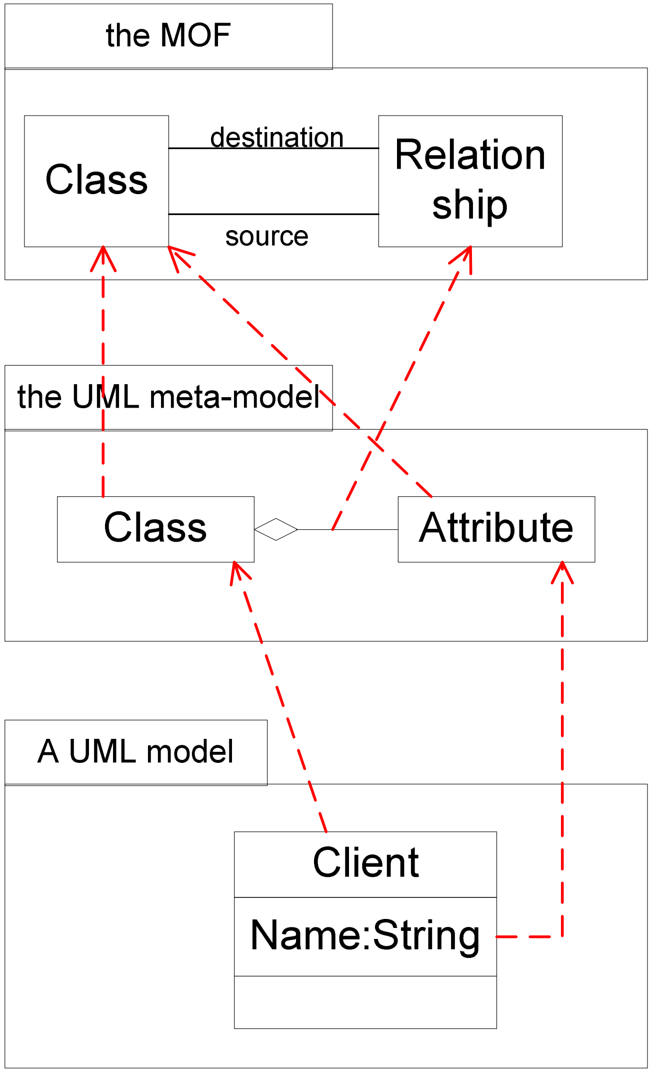
Revision: Unified Modeling Language (in german language)

Model-driven architecture

Modeling of multimedia applications

Context of UML: The OMG Four-Layer-Architecture

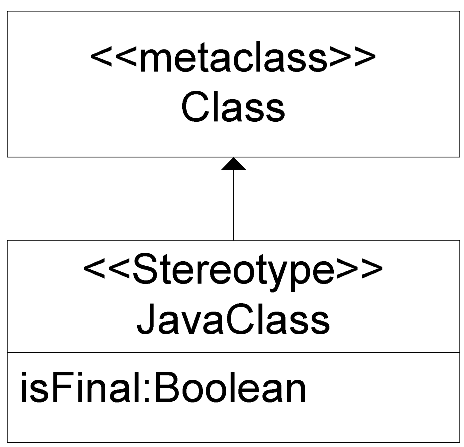
- Idea: the definition of the modeling language is a model itself!
- In context of UML: OMG defined four layers
- Layer M1: UML models (as shown on the previous slides)
- Layer M0: the running application, consisting of concrete instances of the UML model (i.e. objects)
- Layer M2: a model which specifies the UML, so-called *meta-model*. Classes in the metamodel often called *metaclasses*
- Layer M3: a model (*MOF*) which defines all possible metamodels („meta-metamodel") besides UML
- A complete definition of a model requires additional rules and textual explanation!



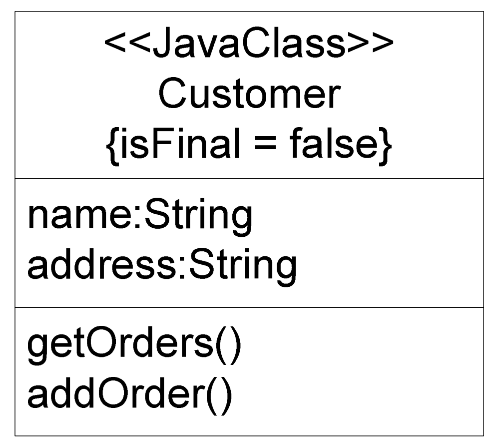
Extension Mechanisms

- **Option 1:** Defining a new metamodel (on layer M1), maybe reusing some UML metaclasses
- **Option 2:** Extending UML using the built-in UML extension mechanism: *Stereotypes*
- A Stereotype defines a new, specific subtype of an existing UML metaclass.
- A Stereotype is represented by its name (in guillemets «») or optionally by its own specific graphical notation.
- Example: Stereotype «JavaClass» to model a class in the programming language Java (which e.g. allows no multi-inheritance)
- A *Profile* (i.e. a specific package) contains a set of stereotypes for a specific purpose (e.g. for a Java-specific model)

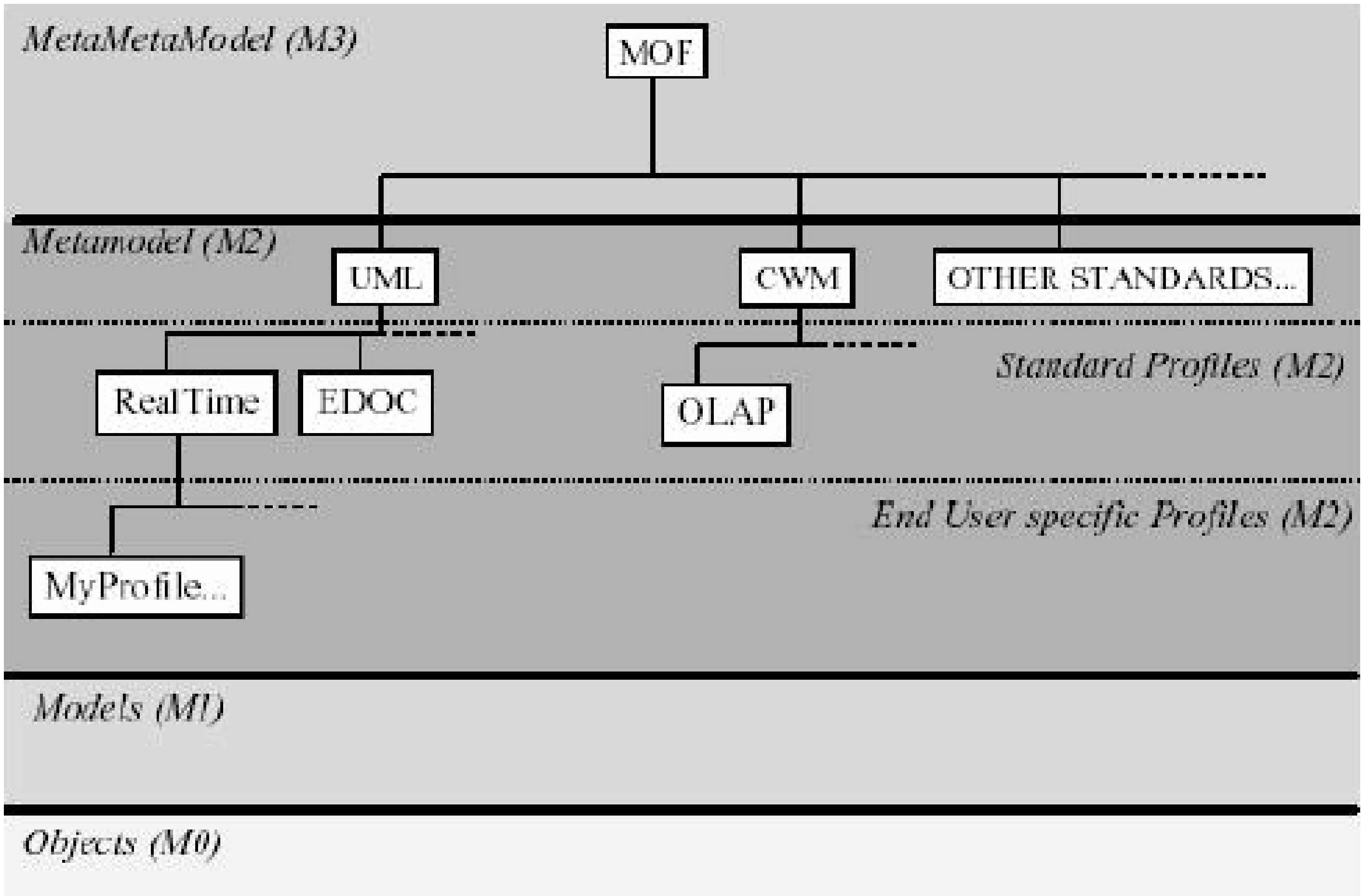
Stereotype definition:



Application of a Stereotype:

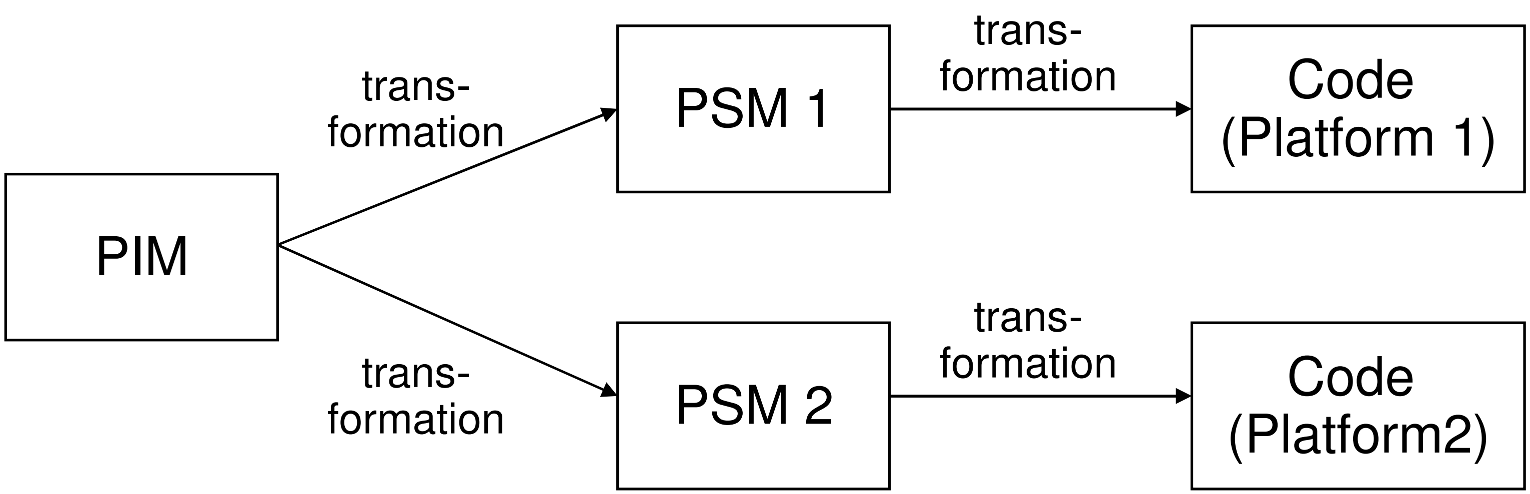


Metamodels and Profiles in the Four-Layer-Architecture



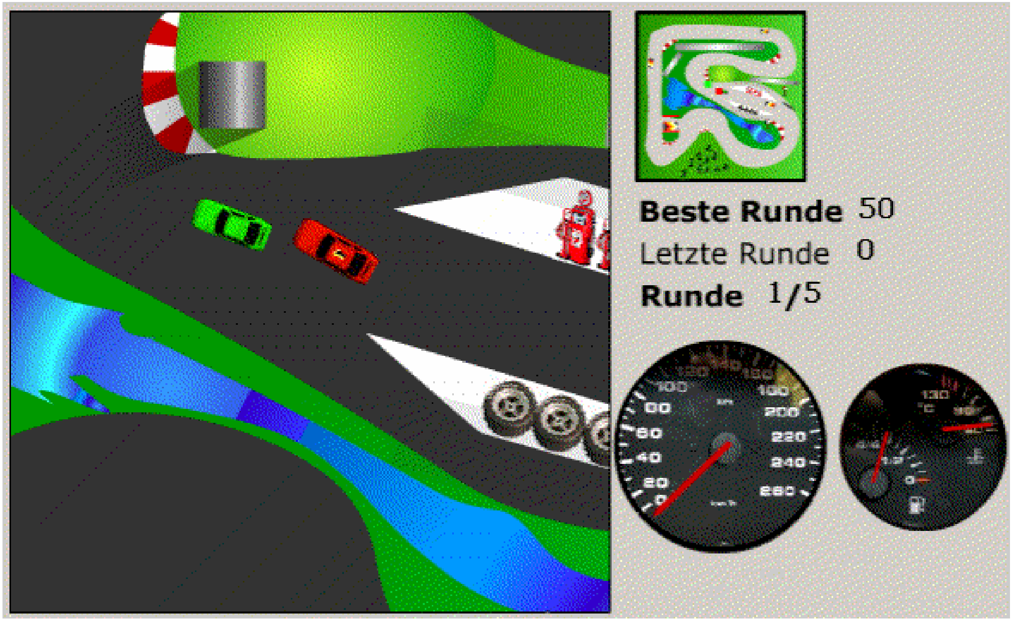
Model-Driven Development

- Idea for *Model-Driven Development* (MDD):
  - Programming on an abstract conceptual level using models
  - Use the models for automatic generation of program code for different target platform
  - Difficult expert knowledge can be put into the code generator
- *Model-Driven Architecture* (MDA): A concrete framework defined by the OMG for the realization of MDD
- Bridging the gap between *platform-independent model* (*PIM*) and *platform-specific implementation* by a platform-specific model (*PSM*)



2 Development process for multimedia projects

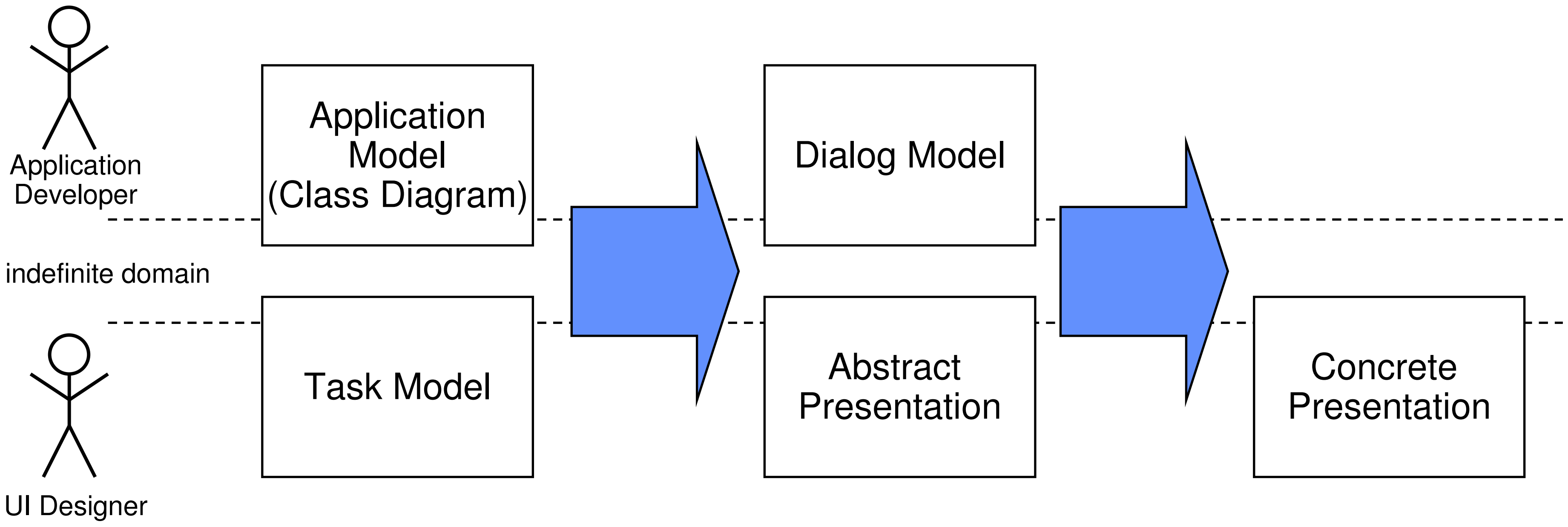
Modeling of Multimedia Applications



- Example: Racing game application
- Specific characteristics of multimedia applications:
  - Integration of media objects like sound, video, animation, 3D graphics, etc.
  - High importance of the user interface
  - High degree of interaction
- Conventional UML not sufficient for modeling multimedia applications

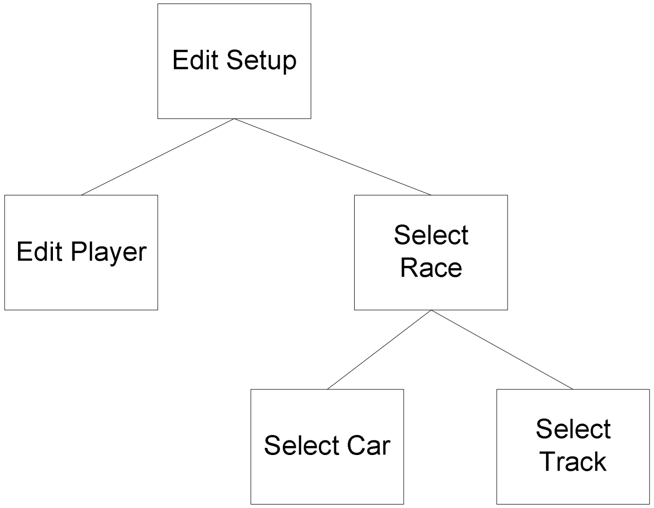
Modeling Interactive User Interfaces

- Standard UML does not cover the user interface aspect
- Extensions of UML for User Interface (UI) Modeling:
  - UML Profile for Interaction Design (*WISDOM*)
  - UML for Interaction (*UMLi*)
  - Profile for Context-Sensitive User Interfaces (*CUP*)



Task Model

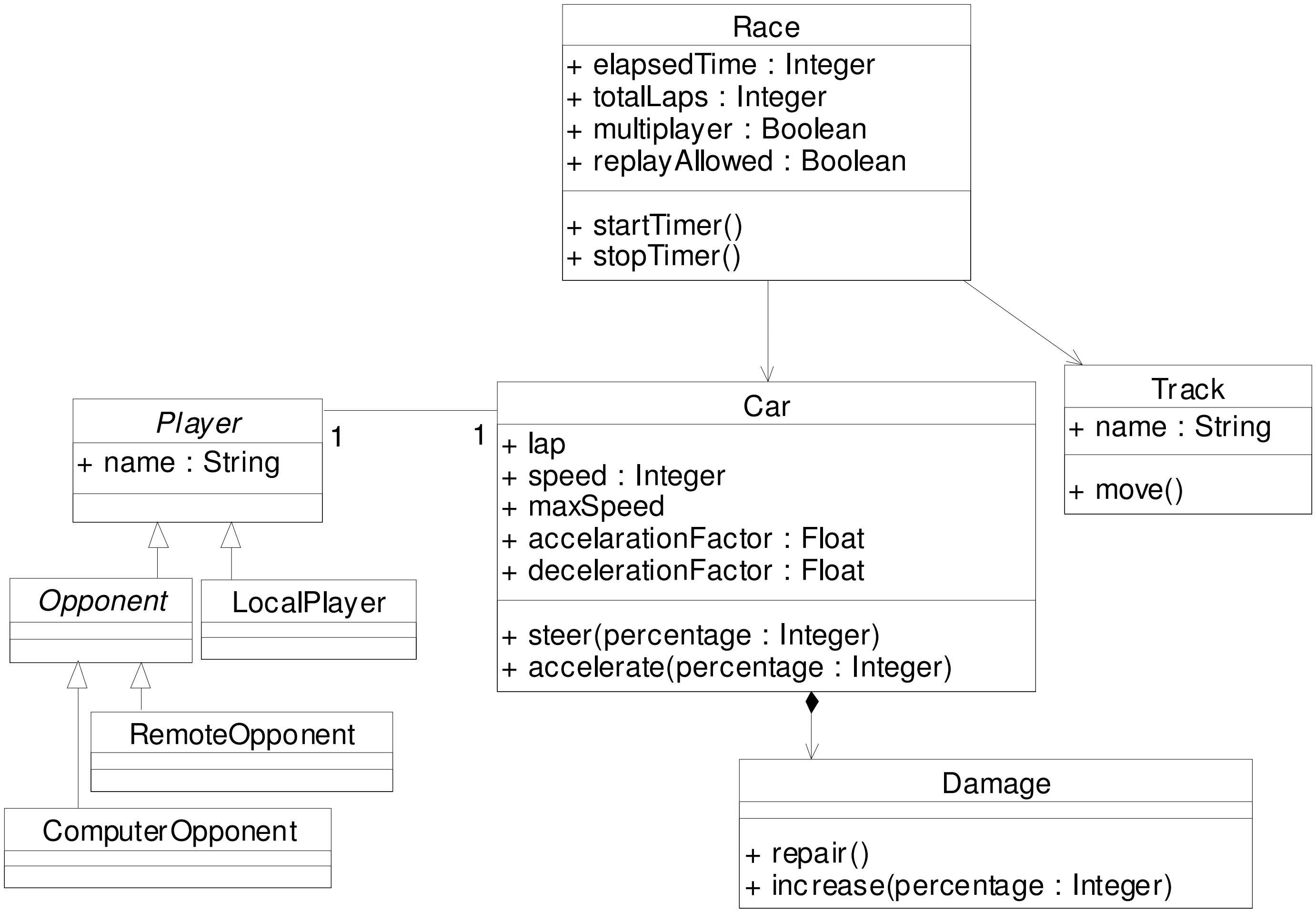
- Modelling the tasks which are performed by the user of the application
- Different approaches:
  - Goals, Operators, Methods and Selection Rules (*GOMS*)
  - Hierarchical Task Analysis (*HTA*)
  - ConcurTaskTrees (*CTT*)
- CTT:
  - frequently used for UI modeling
  - hierarchical decomposition of user tasks which have to be performed to reach a certain goal
  - 4 types of tasks: user tasks (cognitive), interaction tasks, system tasks, abstract tasks
  - tasks linked by temporal operators (e.g. whether tasks are performed sequential or in parallel)



- Task modeling not (directly) supported by UML; adaption of existing diagram types for task modeling:
  - UML use case diagrams (see e.g. UMLi)
  - UML 2.0 Activity Diagrams (see e.g. CUP)

Application Model

- Ordinary UML class diagram

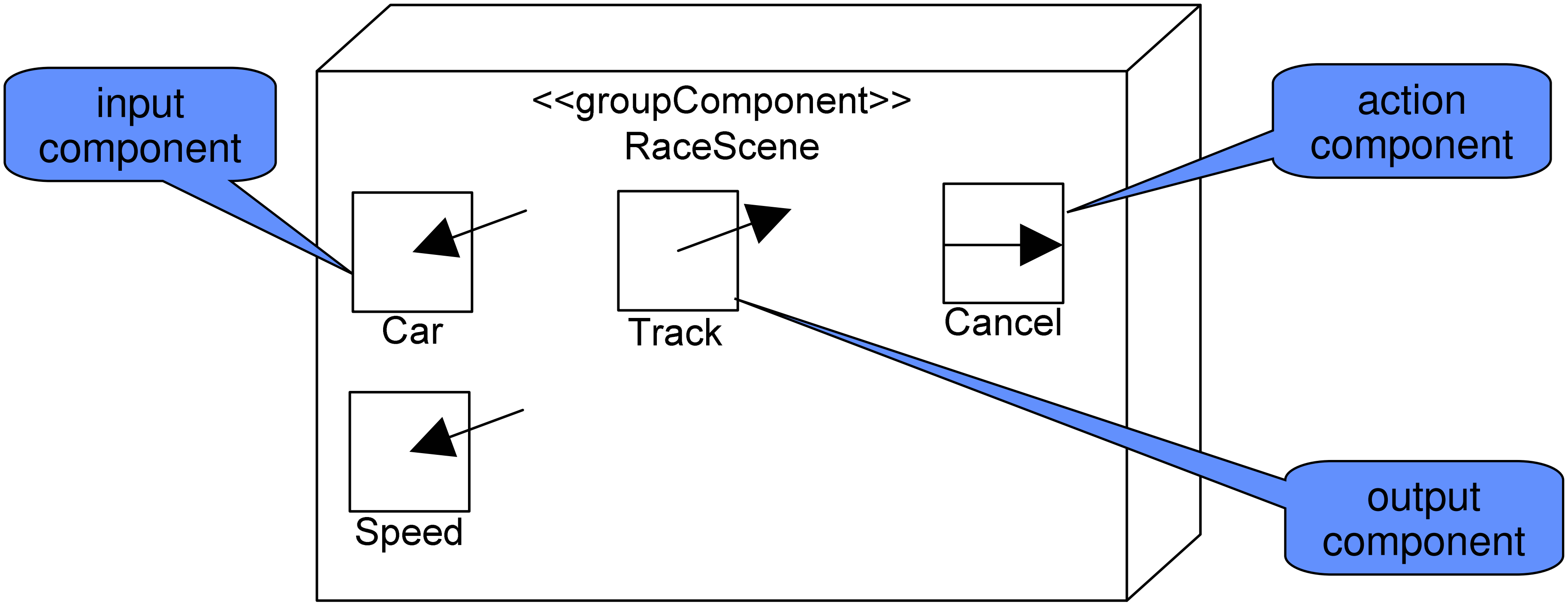


Abstract Presentation

- Abstract structure of the user interface presentation
- In general: three kinds of elements:
  - *Abstract interaction objects* represent low-level tasks, like text input or choosing an element from a set (usually implemented by *widgets* like textfield, combobox, button, etc.)
  - *Information elements* represent information, either static (like a text label) or from the application model (e.g. the value of an attribute)
  - *Presentation units* represent container for the interaction objects and information elements (implemented e.g. by a window)
- In *CUP*:
  - *inputComponents* allows the user to input data
  - *outputComponents* show data to the user without allowing user input
  - *actionComponents* allow the user to trigger actions (e.g. like a button)
  - *groupComponents* group the other components into a logical structure
- Relationships (*select*, *interact*, *trigger*) from UI components to classes from the application model

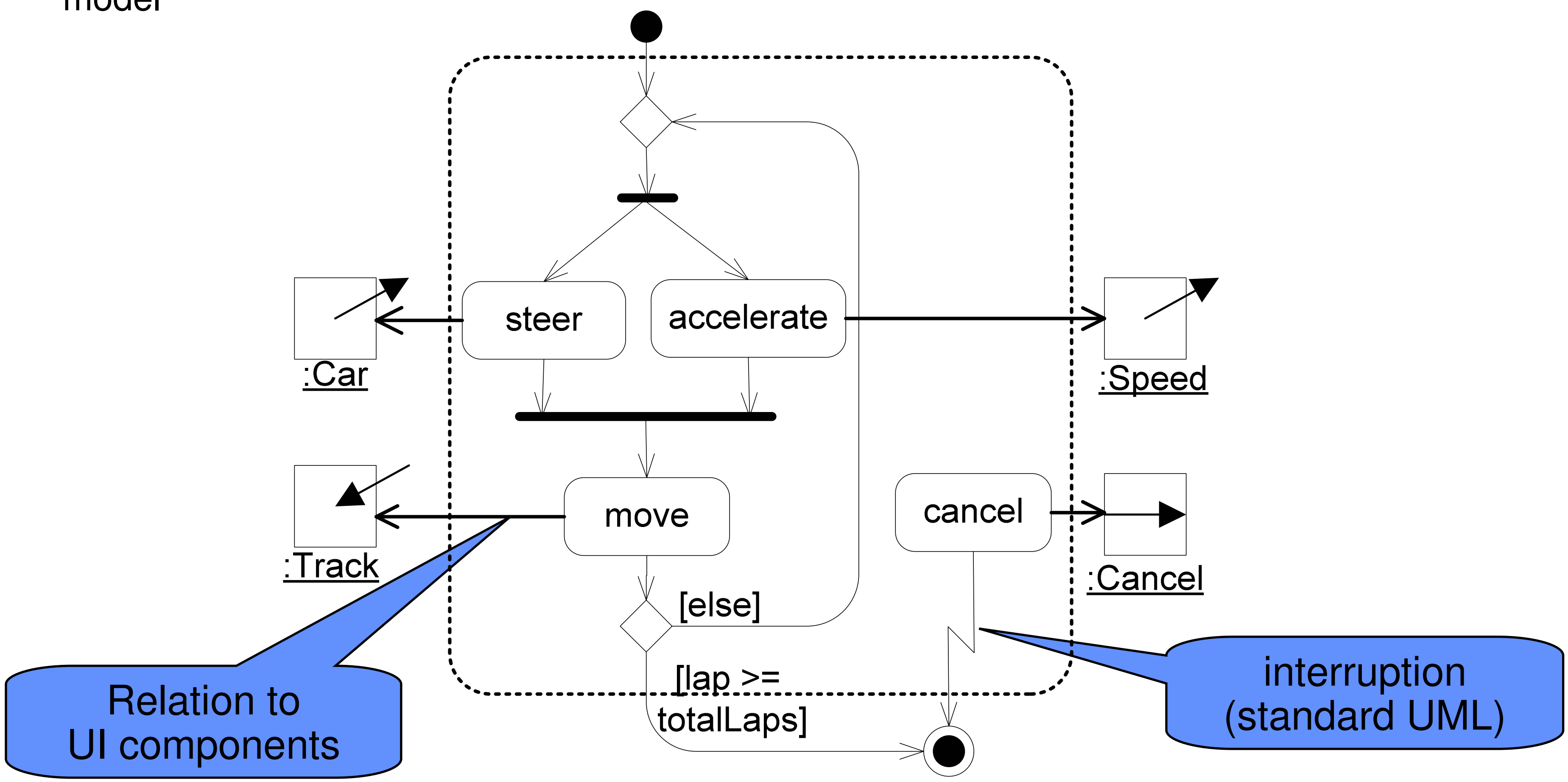
Example: Abstract Presentation for the Race Scene

- New diagram type based on UML class diagram
- Here simple example:
  - the application shows the car, its speed, and the track
  - the user can steer the car and its speed and he can cancel the race



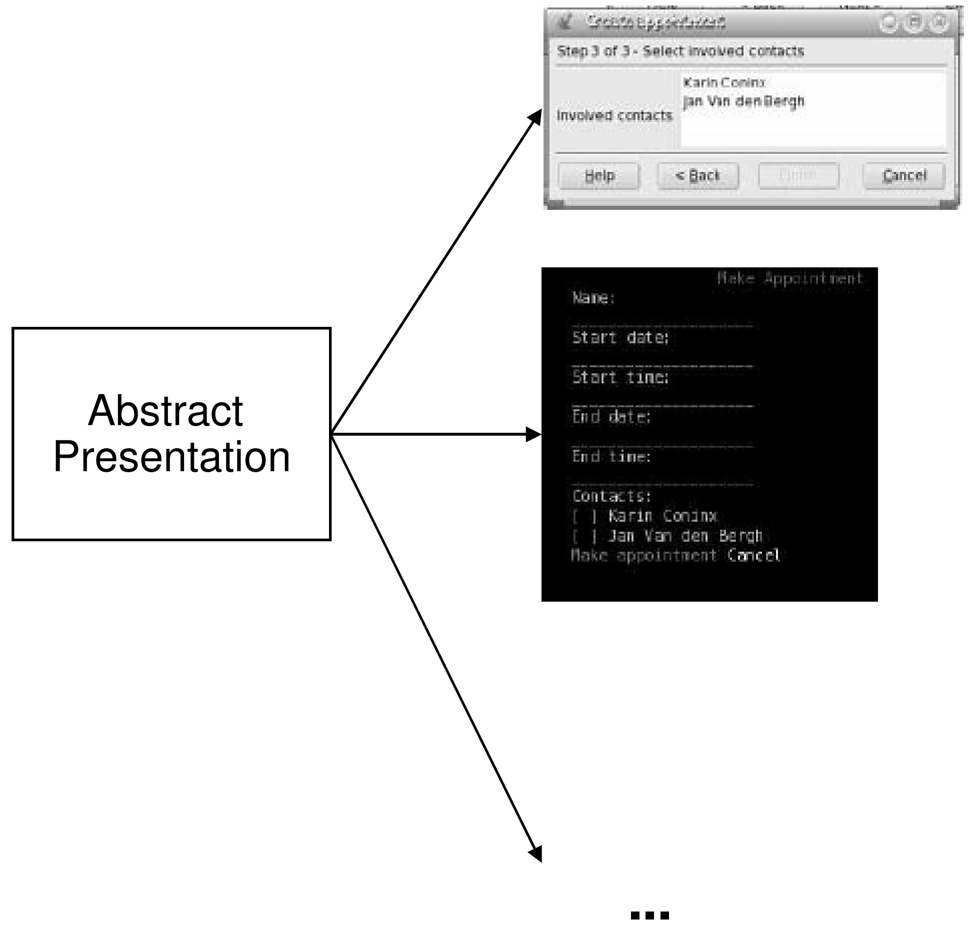
Dialog Model for the Race

- Activity diagram for modeling the behavior of the user interface, i.e. the dialog between the user and the system
- Additional: relationships to the user interface elements from the abstract presentation model



Concrete Presentation

- Specifies the concrete user interface:
  - concrete UI elements, i.e. widgets
  - layout within the goupComponent
  - look of the UI elements and additional adornments
- Often no further distinction between concrete presentation and the final implementation of the UI
- Thus, concrete presentation often better implemented directly in the specific user interface tool (e.g. UI builder)
- Several apporaches exist to partially generate the concrete presentation from abstract presentation
- One abstract presentation can lead to many concrete presentations for different platforms and devices!
- Concrete layout is **not** limited to visual components! OutputComponent can also be realized e.g. by speech output!

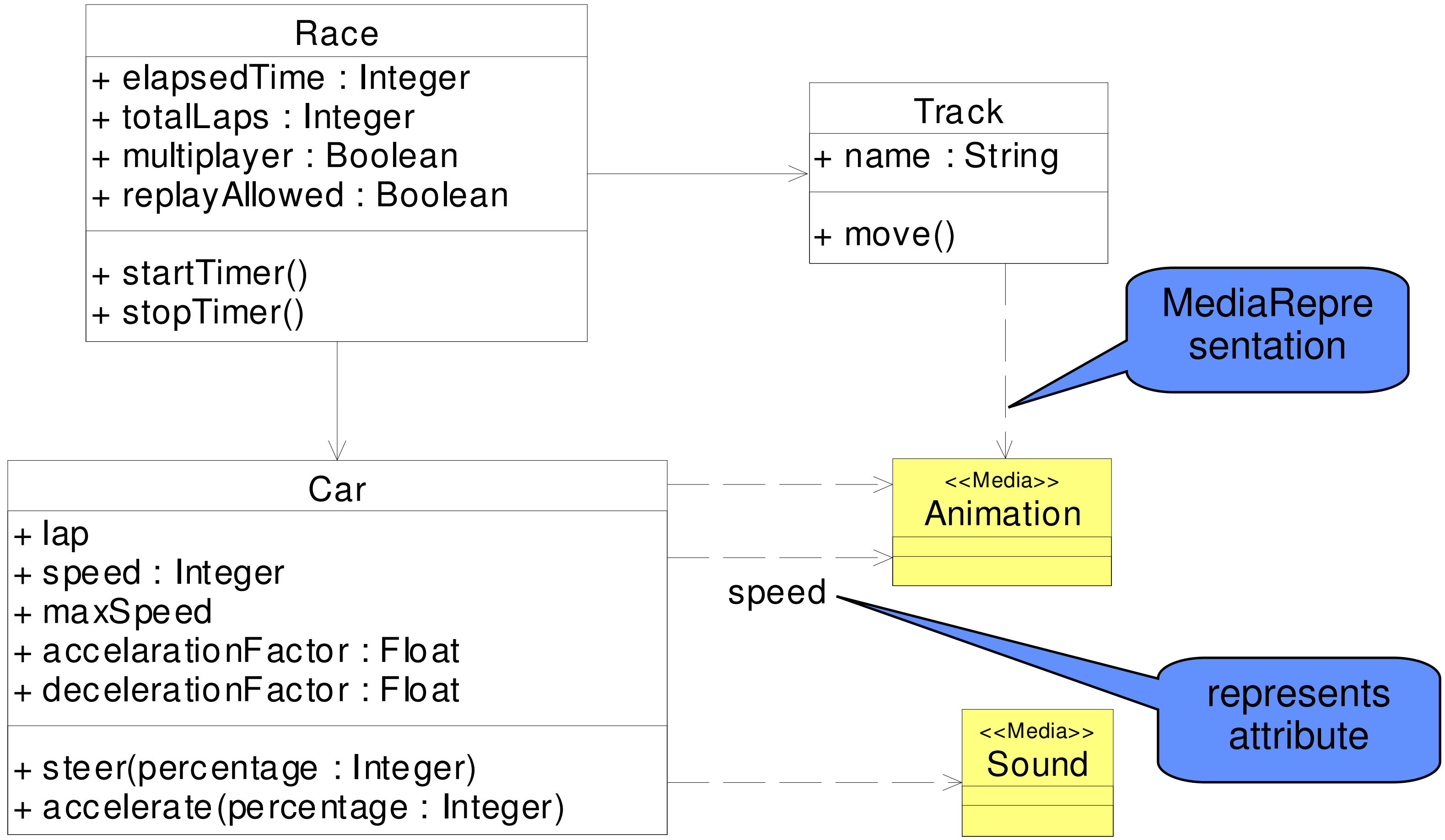


Integration of Media Objects

- Further extension of UML and user interface modeling
- Only few proposal approaches available
- Usage of specific media types is often fix requirement for the application
- Example: for a medical training application, the customer postulates medical organs to be represented by 3D animations
- Consequence:
  - Specification of media objects from start (**not** just during modelling the concrete presentation)
  - Media objects often represent classes (or class attributes) from the application model
- Media objects should be visible and must be controlled by the user, e.g. for a video, it should at least be possible to play the video and probably also to pause, restart, etc.
- Consequence: media objects in the model implicitly represent components providing all common (player-)functionality

Integration in Application Structure

- Extension of UML class diagrams: Media type components and a special kind of relationship (called *media representation*)
- Media representation can also refer a single attribut/operation of the class

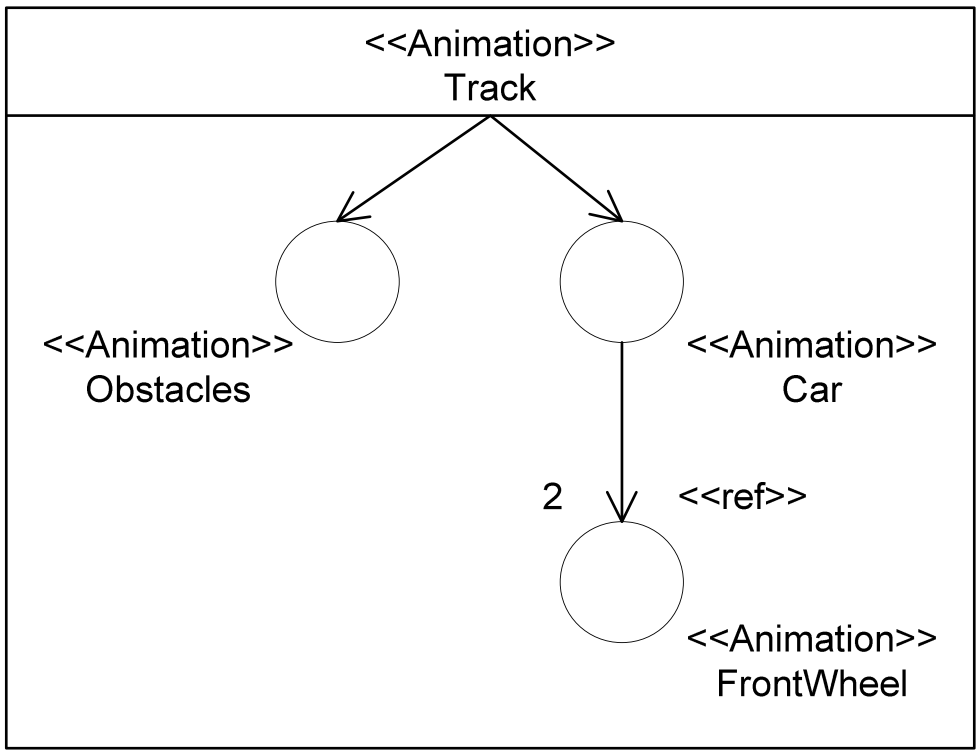


Inner Structure of Media Objects

- Media objects often consist of several (sub-)objects
- Examples:
  - Flash: animations can contain animations themselves
  - 3D graphics: usually consist of different components like primitive objects, transformations, light, etc.
- Inner content often a hierarchy of arbitrary depth
- Program code of a multimedia application often manipulates inner sub-objects of a media object
- Problem: Program code and inner structure of media objects must fit together
  - Media designer: needs to know which parts of a media object have to be designed as independent sub-objects (e.g. because they should be manipulated)
  - Programmer: needs to know how to access the sub-objects
- Consequence: inner structure is a relevant information and to be specified in the model

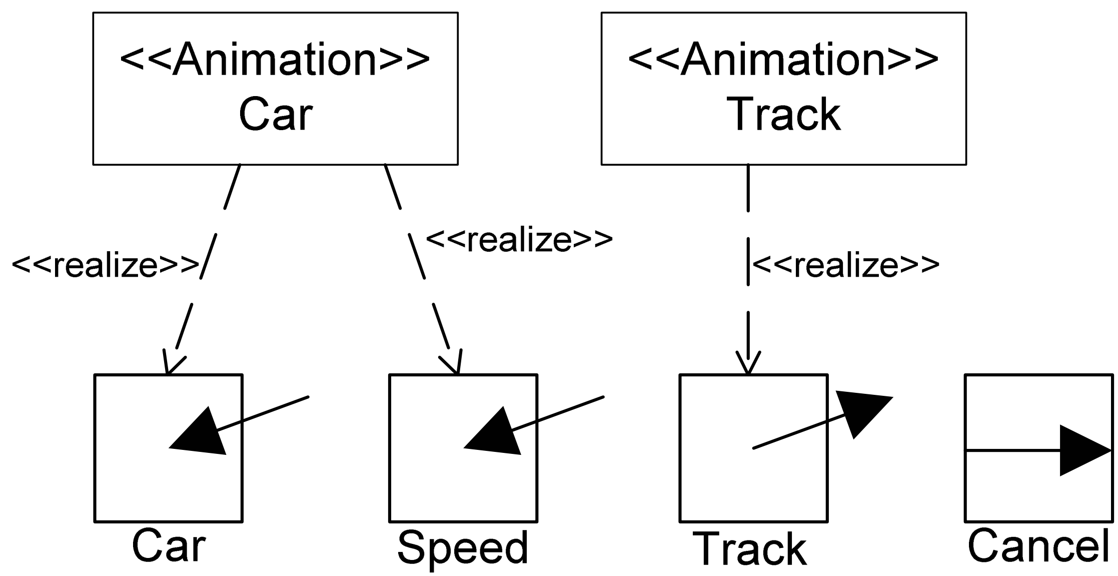
Example: Inner Structure of the Track Animation

- Extension of UML class diagram
- The relationship between Car and FrontWheel is annotated by a multiplicity (‚2‘) showing the number of front wheels
- The keyword *ref* indicates that the two wheels are two references of the same object and thus behave indentically (in contrast to *copy*)

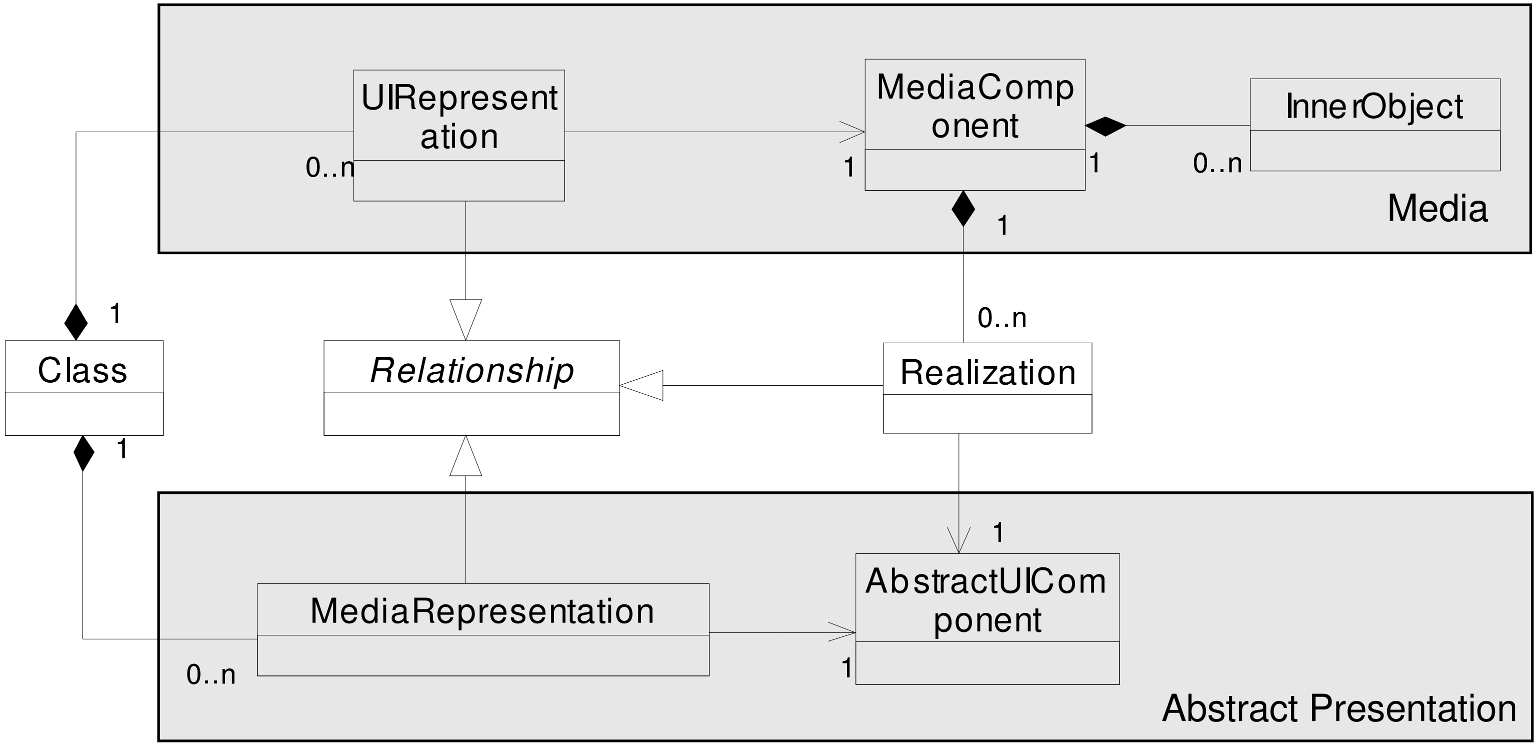


Integration with Dialog Model

- Media objects appear on the user interface
- Consequence: they can act as user interface elements from the abstract presentation model
- As media objects often support complex input and output, they can act as several input, output, and action components at the same time
- Usually, not all user interface elements of a multimedia application are realized by special media objects. There is still a need for conventional UI elements!
- Consequence:
  - Separation between media objects and (conventional) abstract user interface elements
  - Connections between them (relationship *realizes)*



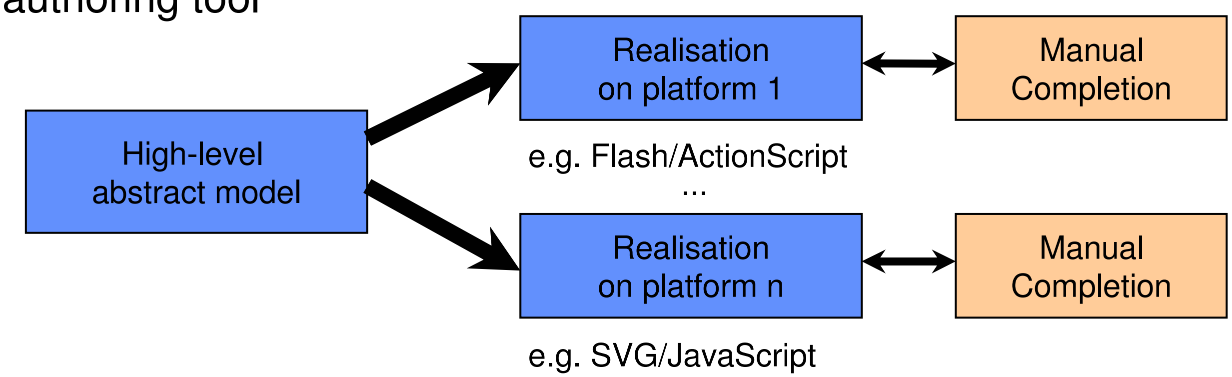
Metamodel Extract



- Shows concepts independent from their notation
- Useful for tool support:
  - Verification of diagrams
  - Existing tools (e.g. Eclipse Modeling Framework) generate code (e.g. as basis for a modeling tool) directly from a metamodel

Tool Support

- Multimedia development: powerful tools required for media design, e.g. authoring tools like Flash
- Consequence: Integration of these tools in the (model-driven) development process
- Strenght of authoring tools:
  - Concrete media objects, graphical design, and layout
  - Application can be executed immediately (e.g. to observe the result of a skript)
- Weakness of authoring tools:
  - Overview on the application parts
  - Structuring the programming code
- Idea:
  - Platform-independent model to develop the overall application and the structure for the programming code
  - Transform PIM into platform specific code-skeletons which should be optimized for the authoring tool
  - All further processing (filling the gaps in the code-skeletons) is done using the authoring tool



Generated Code-Skeletons

- Code-skeletons contain placeholders for the concrete media objects and UI objects (as they are not modelled within the PIM)
- In addition, the implementation of operations (from the class diagram) is not specified in the PIM:
  - it takes much effort to define the complete programming logic within a diagram
  - in many cases, programming logic requires platform-specific features and has to be optimized for the platform
  - in multimedia applications, the specific implementation often has to be figured out by „trail and error“ to get the expected effect (e.g. a specific visual effect on the user interface)

Diagram	**Generated Code**
Class Diagram	ActionScript Classes
	Placeholders for operation bodies
Abstract Presentation	Overall UI structure
	Placeholders (e.g. rectangles) for concrete media objects and UI objects. The placeholders are connected to the required behavior (e.g. event handling code).
Dialog	ActionScript Code for the application‘s overall behavior

Synchronization of Model and Code

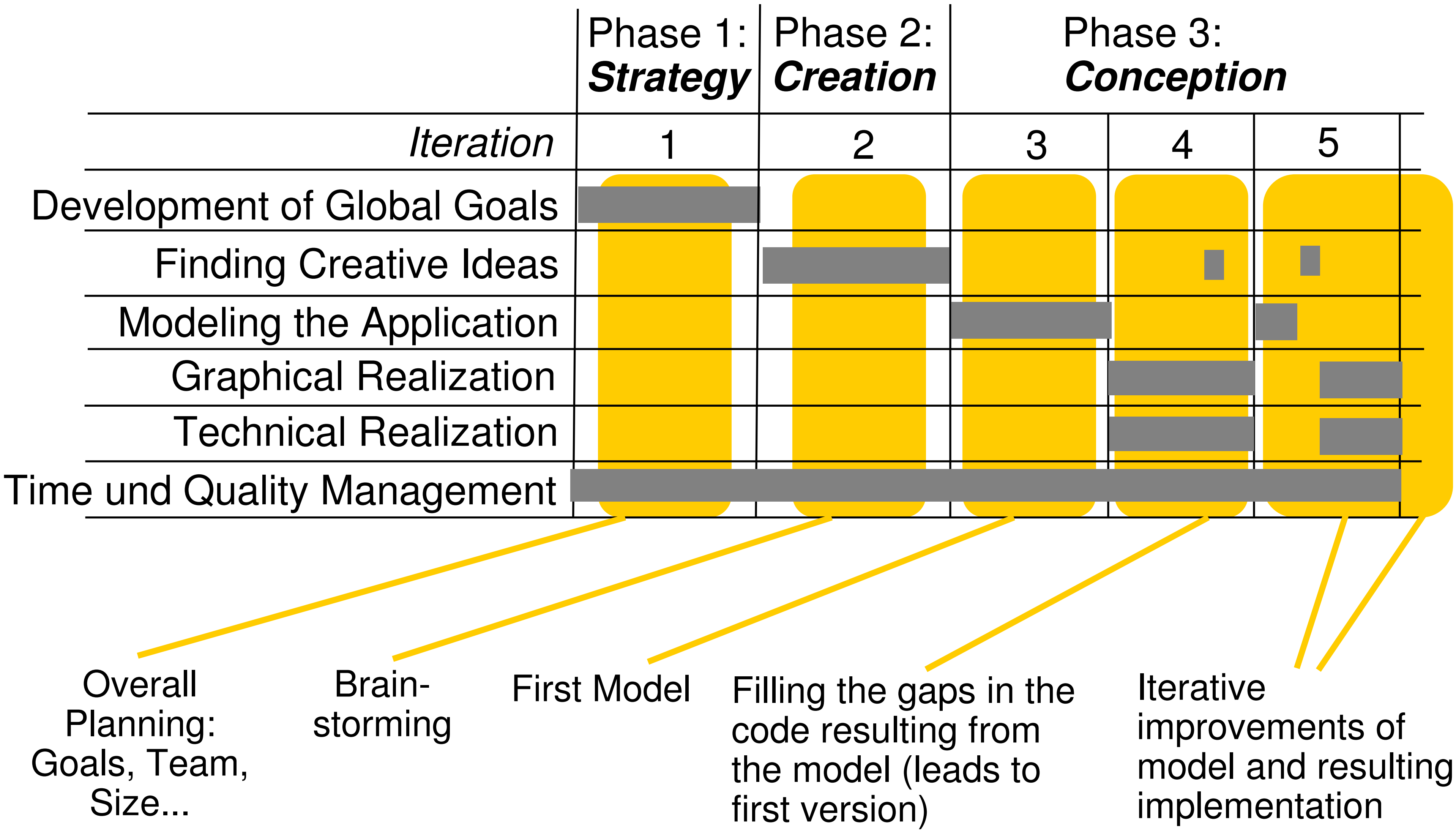
- If the generated code is modified in the authoring tool, then the platform-independent model is deprecated
- Consequence: Model has to be updated when changes in the generated code occur
- If some placeholders are already filled out in the authoring tool and then a change in the model is performed (e.g. change fo the class diagram) the next code generation will generate empty placeholders again
- Consequence: code generator must only overwrite the old generated code but must not touch information added by the developer/authoring tool
- Possible solution: *Round-Trip-Engineering*
  – Authoring tool observes changes in the generated code and provides a command which updates the model (e.g. a plug-in for the authoring tool)
  – Modeling tool observes changes in the model and performs only those changes in the generated code.
- Advanced solution: integration of modeling tool and authoring tool:
  – direct synchronization of model and code
  – model and code act as two different views on one system
  – developer switches between the two views whenever he wants

Model-Driven Development vs. Extreme Programming

XP	MDD
Implementation directly after requirement analysis	Modeling phase between requirement analysis and implementation
Focus restricted on the next steps	Focus from start on the end product
Restricted to small teams	No specific team size
Collective ownership of code; team members work on several (all) parts of the system	Model can be used to divide system and distribute responsibility on the team members
Documentation system based on the code itself; always up-to-date	Documentation of the system based on abstract diagrams; good understanding
Design as simple as possible	Use abstract model to optimize structure from start

- However: Executable models (i.e. containing the **complete** information about the application) also allow applying the XP principles (*agile MDA)*

Integration in the Development Process



Current Research at Media Informatics Group

- For the practical work for this lecture:
  - Option: Use model-driven approach
  - Drawback: No tools for automatic code-generation available
  - Solution:
    - » One modeling phase to achieve structure of the application
    - » Manual code generation according to this lecture
    - » Afterwards work only in authoring tool (no iterative updates of the model)
  - Tutor will give support for modeling
  - Reduced requirements on the application
- Topics for project work or diploma thesis:
  - Tool support for the model-driven development
  - Application and/or evaluation of the model-driven development process for multimedia applications
  - Results will be applied in future editions of this lecture
  - Other problems: e.g. testing for Flash