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

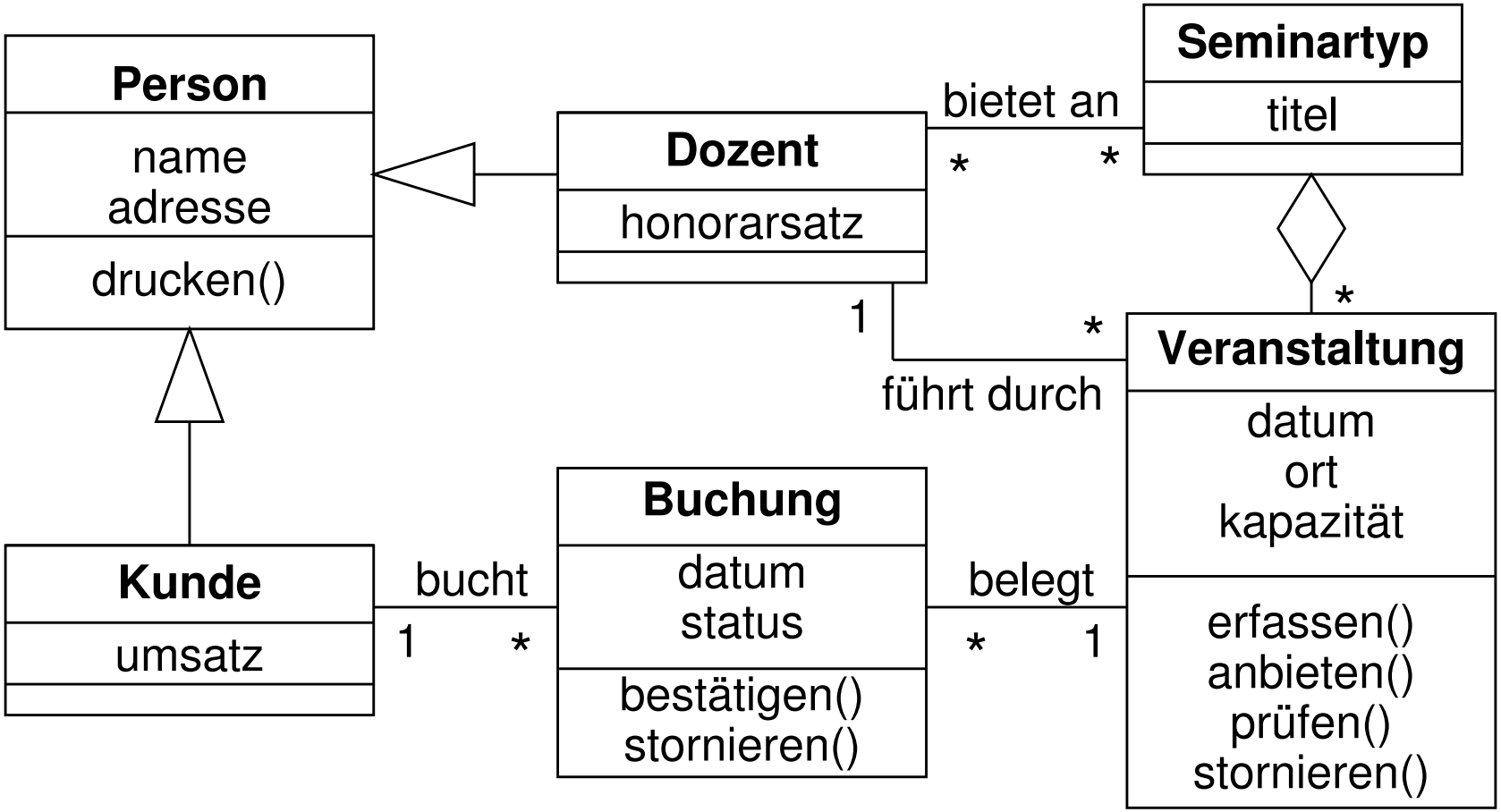
- Models in the software development process
- Specification of modeling languages
- MML: A modeling language for multimedia applications

Models

- How to denote the concepts found during analysis and design phase of the software development process?
  a) Non-formal: e.g. in natural language.
     - Pro: easy to read and write
     - Contra: large descriptions, often inconsistent and ambiguous
  b) Formal: e.g. with mathematical formulas
     - Pro: formal proof of correctness and consistency possible
     - Contra: requires expert knowledge
  c) Semi-formal: e.g. graphical models
     - Compromise between both
- Model: ‘a simplified image of a system’
- Different views on a system, e.g. structure vs. behavior
- Different levels of abstraction, e.g. use case vs. program flow
- Notation: for software development often graphical models (e.g. UML). Advantage: compact, (relatively) easy to understand

Example: Unified Modeling Language

- Specified by Object Management Group (OMG)
- Unifies 13 different types of diagrams (version 2.0)
- Example here: Class diagram

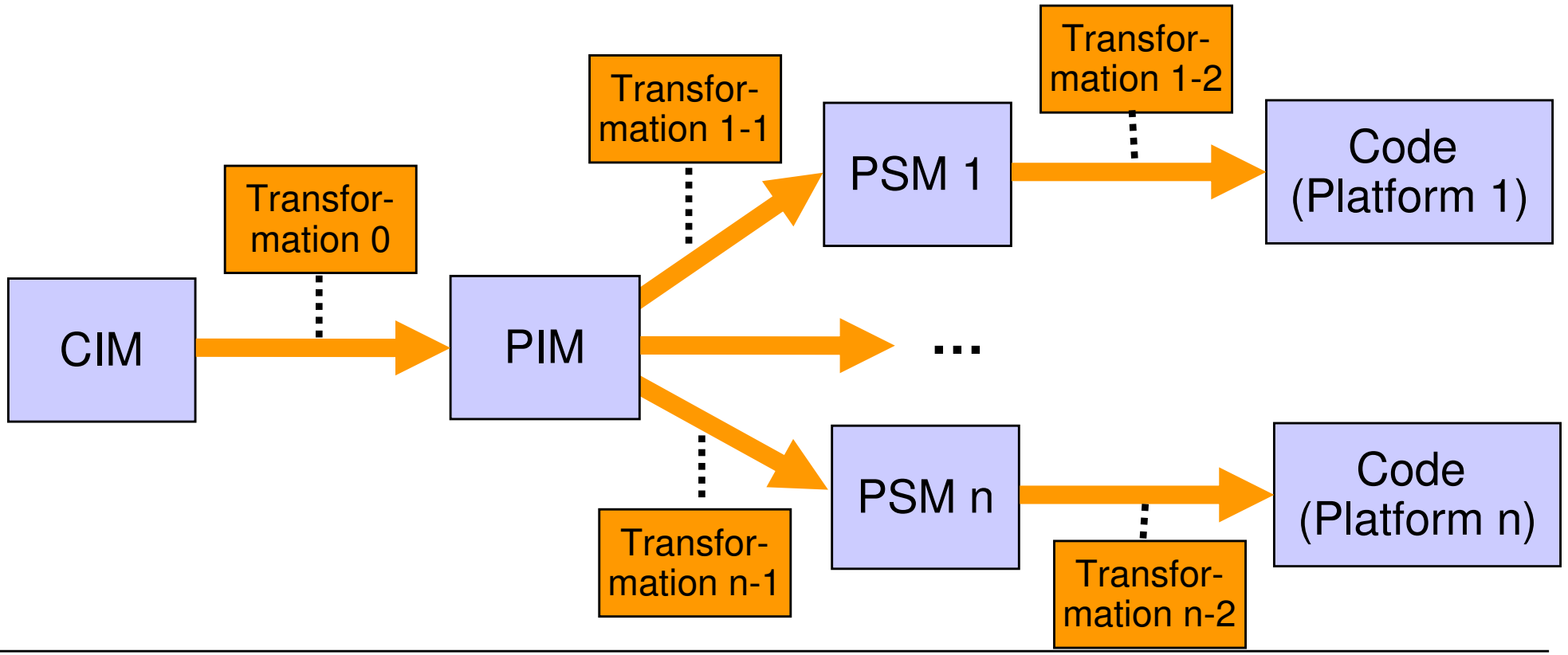


Model-Driven Development

- *Model-Driven Development* (often referred to as *Model-Driven Engineering*): Development process with models as core assets
- Idea:
  - 'Programming' on an abstract conceptual level using models
  - Implementation code is generated automatically from models
  - Expert knowledge about implementation details is put into the code generator
- Ideal case:
  - Complete development process is covered by models
  - No manual completion of code required; code is completely represented by models (on the required level of abstraction)
- Requirements:
  - Various models available to cover development process:
    - » Different levels of abstraction during development
    - » Different views on the system to cover all aspects of the system
  - Transformations (mappings) between the models
    - » Forward, to derive more concrete models from earlier models in a consistent way and to increase efficiency
    - » Backwards, to allow iterations (discussed later)
- Transformations specified explicitly and treated as assets of their own
  - Customizable

Model-Driven Architecture

- *Model-Driven Architecture* (MDA): A concrete framework defined by the OMG for the realization of MDD
- CIM: Computation independent model
- PIM: Platform independent model
- PSM: Platform specific model



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

Models in the software development process

Specification of modeling languages

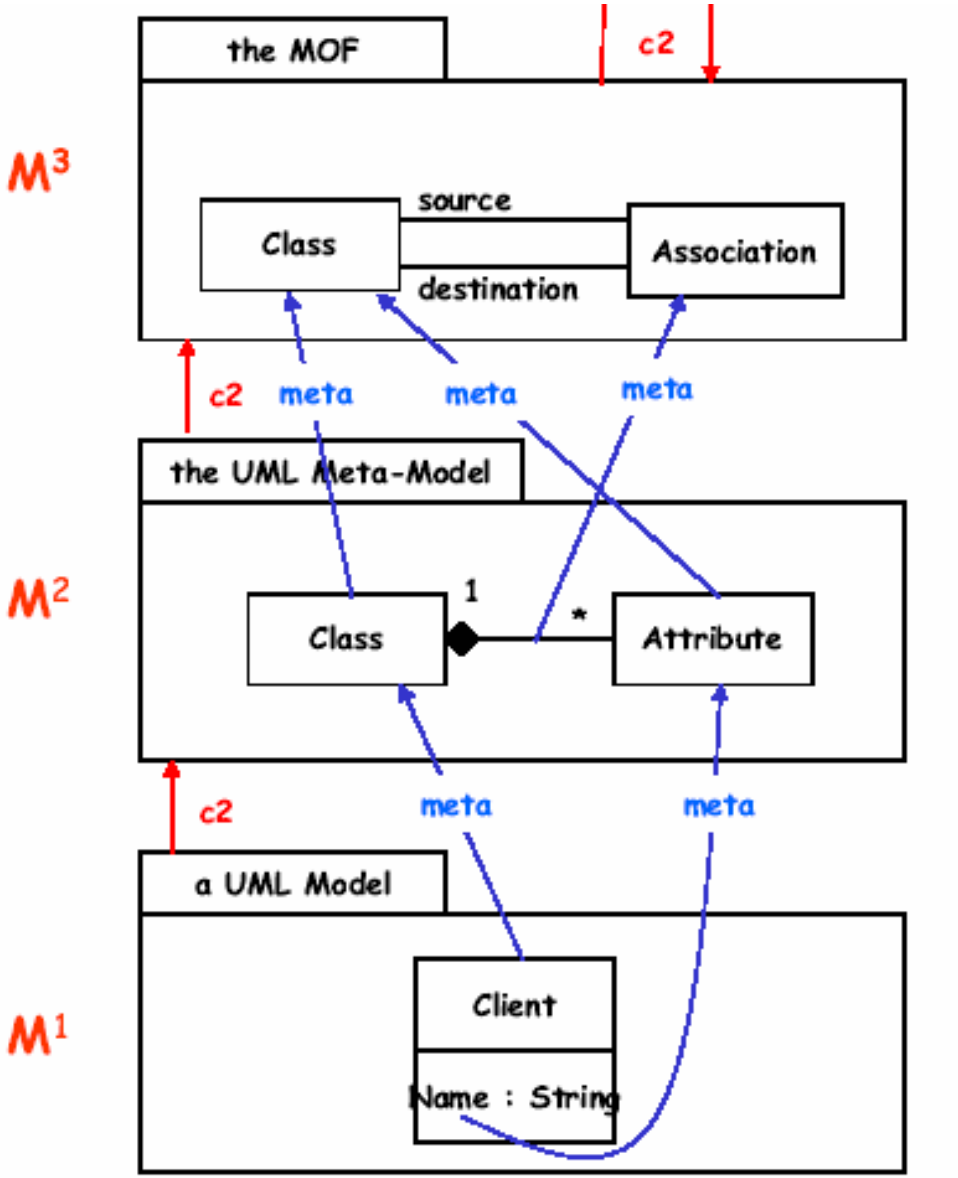
MML: A modeling language for multimedia applications

Customization of Modeling Languages

- Model-driven development requires a set of models customized for the different development tasks
- Aspects of the required modeling language:
  - Application domain, e.g. business application or multimedia application
  - Level of abstraction, e.g. for analysis level or implementation level
  - On low abstraction levels: target platform
    - » Implementation technology, e.g. programming languages or frameworks like Enterprise Java Beans
    - » Device, e.g. mobile phone or PDA
- Plain UML does not always address all these aspects in an optimized way => definition of customized languages
- How to define a (graphical) modeling language?
  - Many possible options: Context-free grammars, EBNF, …
  - Idea in modeling community: *Define the modeling language by a model itself!*
- *Metamodel*: model which specifies a modeling language. Contained elements often called *metaclasses*
- OMG defined Four-Layer-Architecture as framework for the definition of metamodels and models
  - Allows compliant models, tools and interchange formats

The OMG Four-Layer-Architecture

- Layer M1: models, e.g. conventional UML model (or any other customized model)
- Layer M0: the running application, consisting of concrete instances of the model (i.e. objects which are instances from classes of an UML class diagram)
- Layer M2: metamodel, e.g. the UML metamodel which specifies the UML
- Layer M3: meta-metamodel, defined by OMG, called *MOF* (meta object facility)
- Metamodel defines only 'abstract syntax' of a modeling language
- A complete definition of a modeling language requires additional rules (*well-formedness rules*), and a definition of the semantics and the notation of the modeling elements

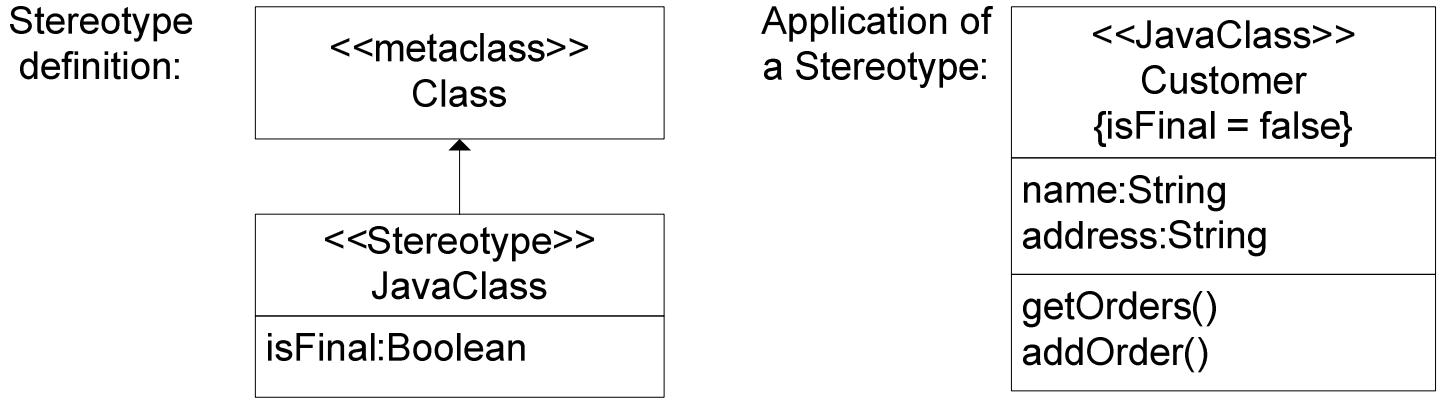


Options for the Specification of Modeling Languages in the OMG Four-Layer Architecture

1. Specification of a new, independent metamodel from scratch
   - E.g. using MOF or similar
   - Pro: no restrictions for optimized customization of the language
   - Contra: much effort, difficult for large modeling languages
2. Customization or extension of the UML metamodel
   - Adding metaclasses to the UML metamodel
   - Pro: UML metamodel well-defined by experts and widely known
   - Contra: might lead to metamodels which do not optimally fit to the own problem (e.g. too large or to few difference from UML)
3. Specification of an extension of UML using the built-in UML extension mechanisms
   - A *UML Profile* is kind of “plug-in“ to customize the UML (see next slide)
   - Pro: UML tools with UML profile support can be used as modeling tool for the customized modeling language
   - Contra: Limited customization, only specialization of existing UML metaclasses

UML Built-in Extension Mechanisms

- A *Stereotype* defines a new, specific subtype of an existing UML metaclass.
- Properties of a stereotype:
  - Name
  - Base class, i.e. the UML class which is extended through the stereotype
  - Attributes; primitive types or references to other metaclasses
  - Optionally a graphical icon which represents the stereotype in a model
- Application: in a model a stereotype is denoted like ist base class, branded with the name of the stereotypes (in guillemets «») and an optional icon.
- Example: Stereotype «JavaClass» to model a class in the programming language Java (difference to plain UML e.g.: no multi-inheritance, property *final*)
- A *Profile* (i.e. a specific package) contains a set of stereotypes for a specific purpose (e.g. for a Java-specific model)



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

Models in the software development process

Specification of modeling languages

MML: A modeling language for multimedia applications

Specific Challenges in Multimedia Development

General goals:

- Efficient and well-structured development methods
- Application of software engineering principles

Main challenges (in contrast to conventional applications):

1. Importance of creative design tasks
   - ⇒Multimedia authoring tools which are oriented towards creative design aspects (e.g. allow visual composition of the application)
   - ⇒Problem: Low support for software design phase and for structuring the application
2. Requires knowledge about media design, user interface design, and software design
   - ⇒Different developer groups involved in development process
   - ⇒Problem: Integration, e.g. by clearly specified interfaces

Solution Approach

Derived from conventional software development:

- Design phase using an abstract visual model of the application (e.g. Unified Modeling Language)
- Generation of application code (skeletons) from the models (Model Driven Development)

Resulting benefits for multimedia applications:

- Specification of platform independent concepts of the application
- Better structured applications
- Integration of different developer groups
- Concrete realization can still be done in authoring tool with visual support for creative design tasks

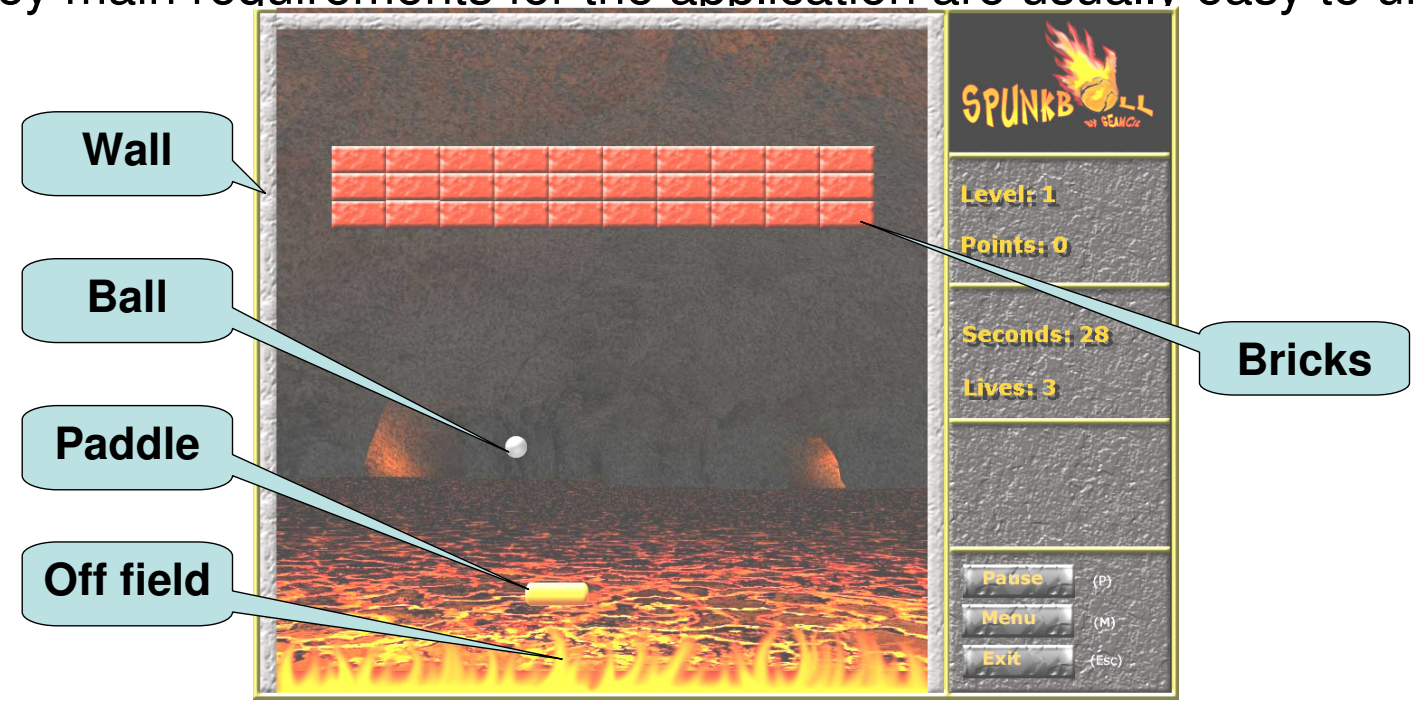
Modeling of Multimedia Applications

- 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
- Plain UML lacks of sufficient support for these characteristics
- Several modeling approaches (mainly in research):
  - Modeling of User Interfaces (e.g. UsiXML, CanonSketch, UMLi, …)
  - Modeling the integration of media objects (e.g. OMMMA)

- Example (shown here): *Multimedia Modeling Language* (*MML*):
- Modeling language, based on UML 2.0, customized for modeling interactive multimedia applications
- Specified as metamodel which extends the UML metamodel
- Aims to integrate software design, user interface design, and media design
- Four kinds of diagrams:
  - Application structure
  - Scenes
  - Abstract User Interface
  - Interaction

Example Application: Break Out Game

- Not all kind of gaming applications are expected to be developed using a modeling language like MML (e.g. high-end 3D games)
- But: (Smaller) games are a good example to show the modeling language as
  - They traditionally make intensive use of media objects, interaction and complex user interfaces
  - They main requirements for the application are usually easy to understand



Application Structure Diagram

Motivation:

- Structure of application logic in terms of a domain model analogous to conventional applications
- In addition: media components as core assets of the application as
  - Usage of specific media types is often a core requirement for the application
  - Provision of media objects can be a appreciable part of the development process
- Integration of media components and application logic
  - Can require a specific inner structure of a media component

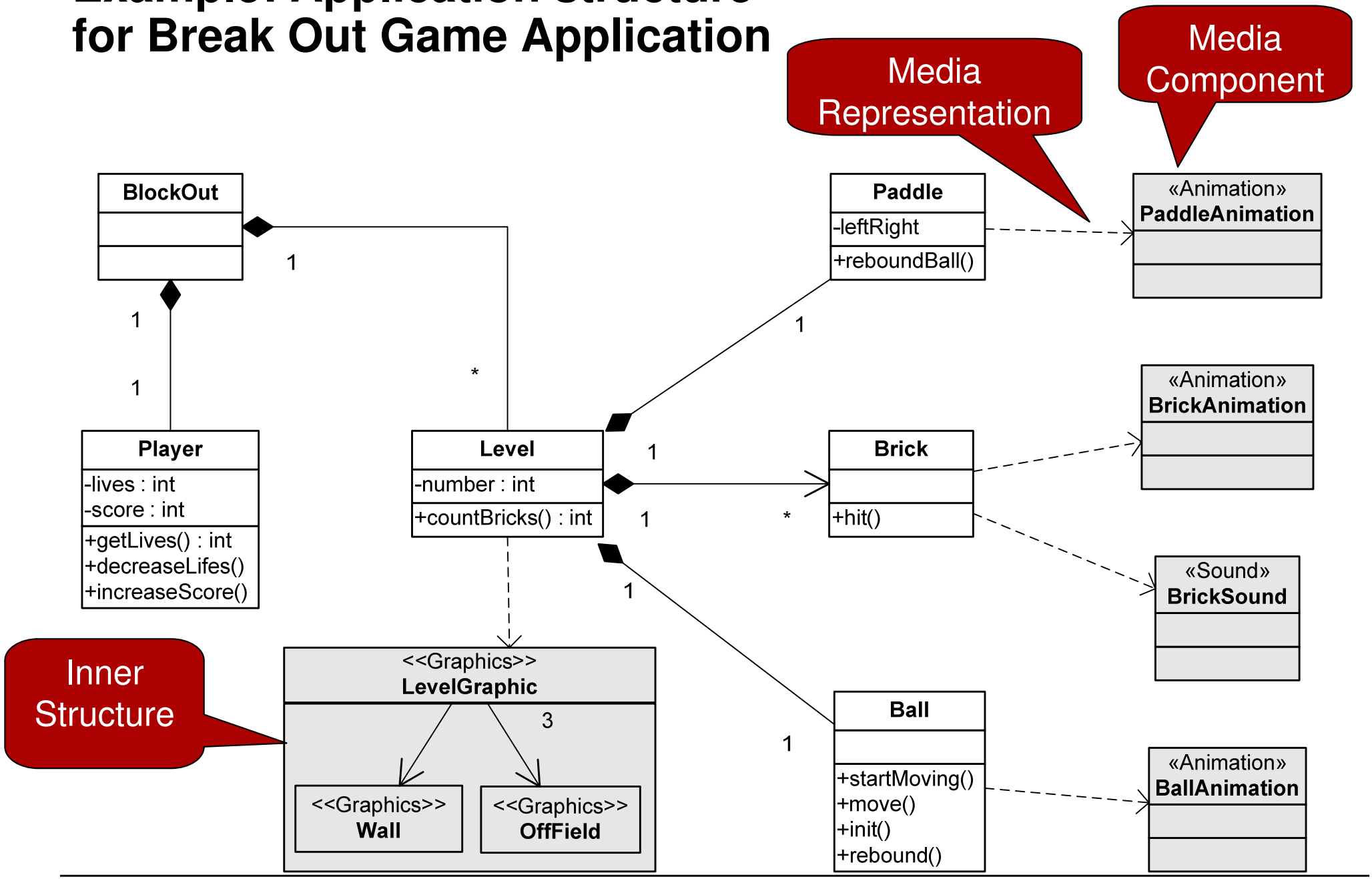
Notation:

- UML class diagram, extended with elements for media components

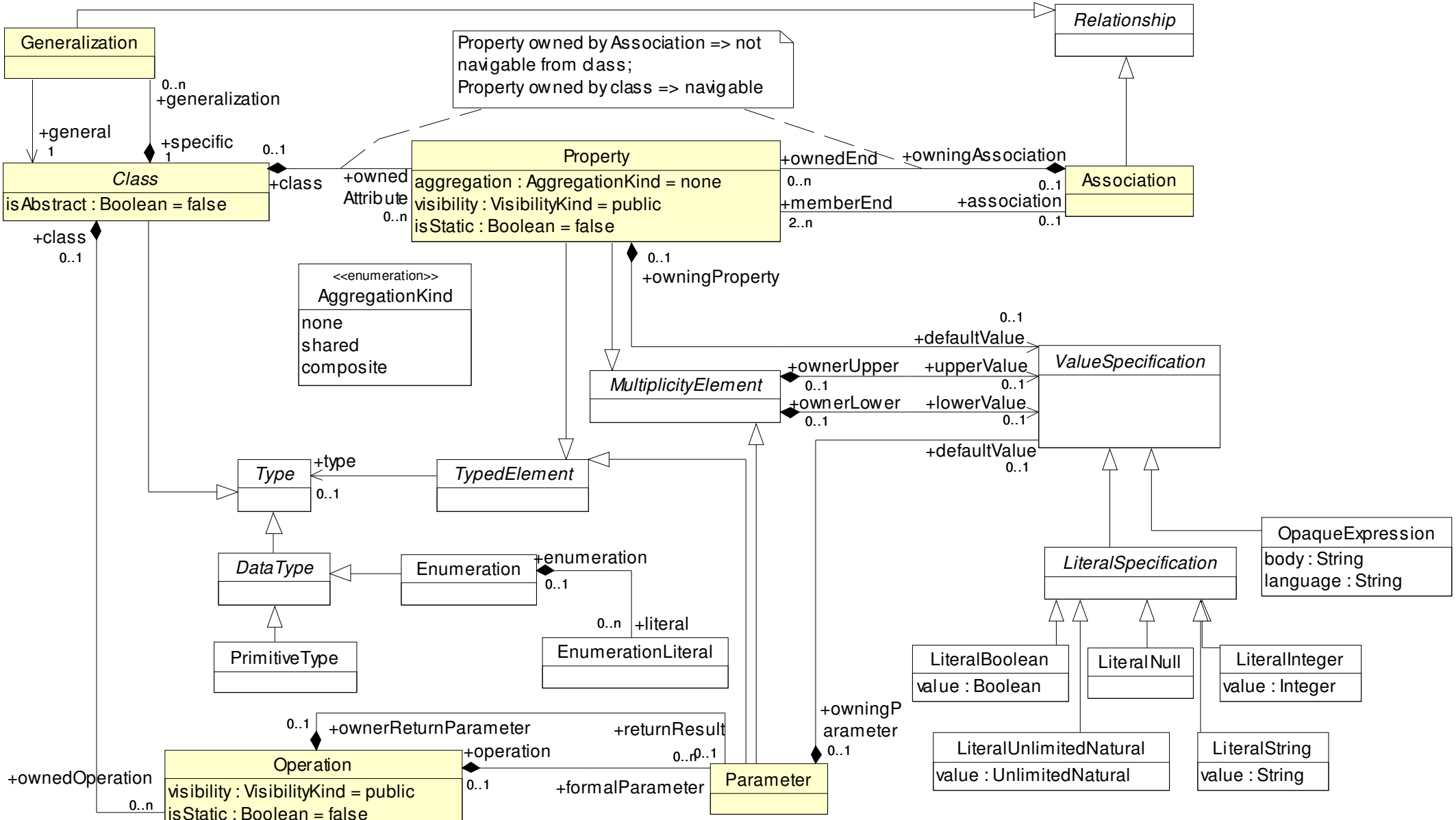
Specific model elements:

- Domain model classes referred to as *application entities* to distinguish from other kinds of classes
- Media components representing a specific media type, like *Animation2D*, *Animation3D*, *Audio*, *Video, …*
- *Media Representation*: Relationship which specifies that an application entity is represented by a media component
  - Multiplicities to specify relationships other than 1:1
- Inner structure of media components denoted as a hierarchical tree (similar to a scene graph for 3D applications) within the media component
  - Edges can be branded with a multiplicity

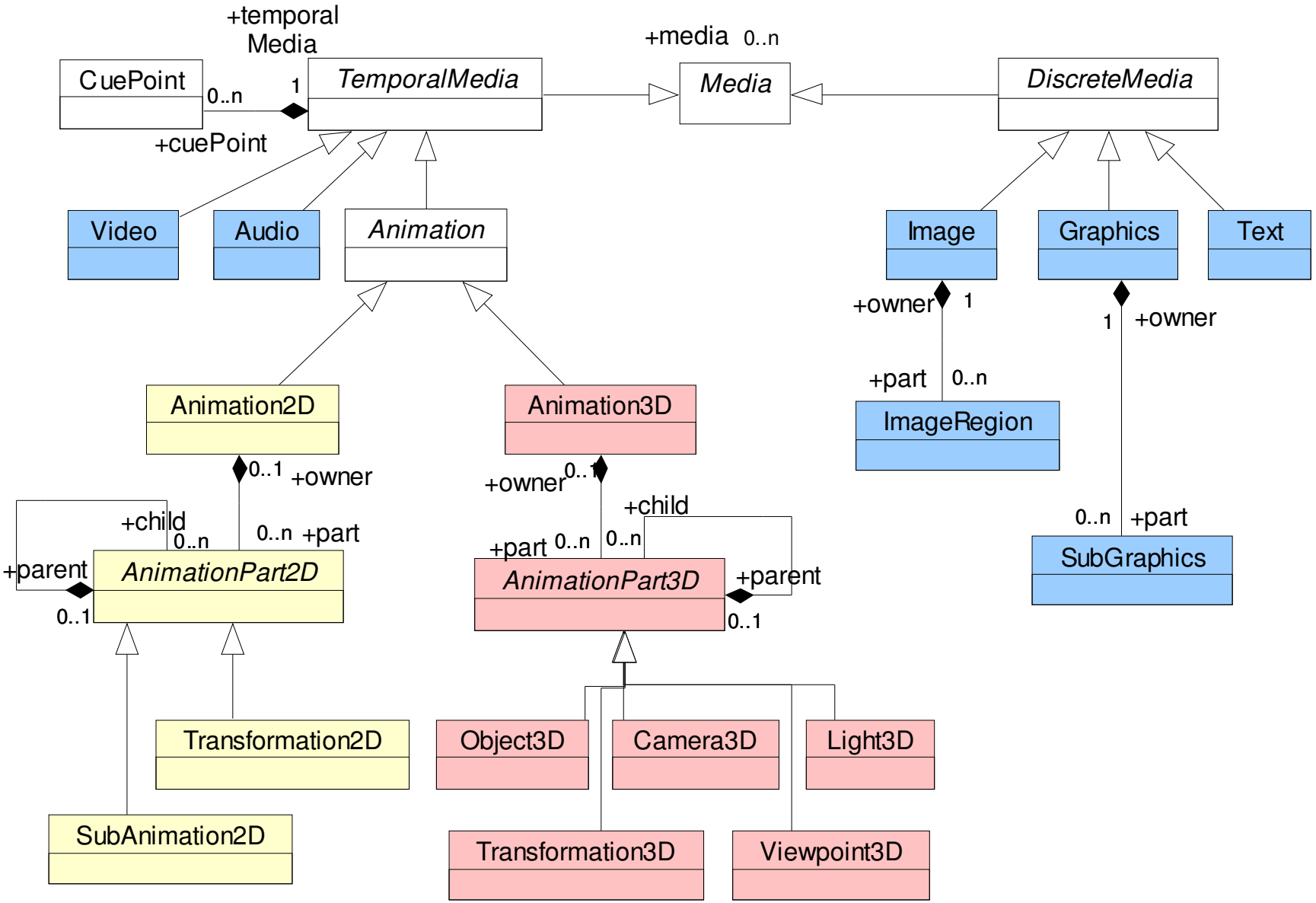
Example: Application structure for Break Out Game Application



Metamodel for class diagram (extract)



Metamodel for media components



Scene Diagram

Motivation:

- Structure of the overall behavior
- Captures ideas e.g. from Storyboards

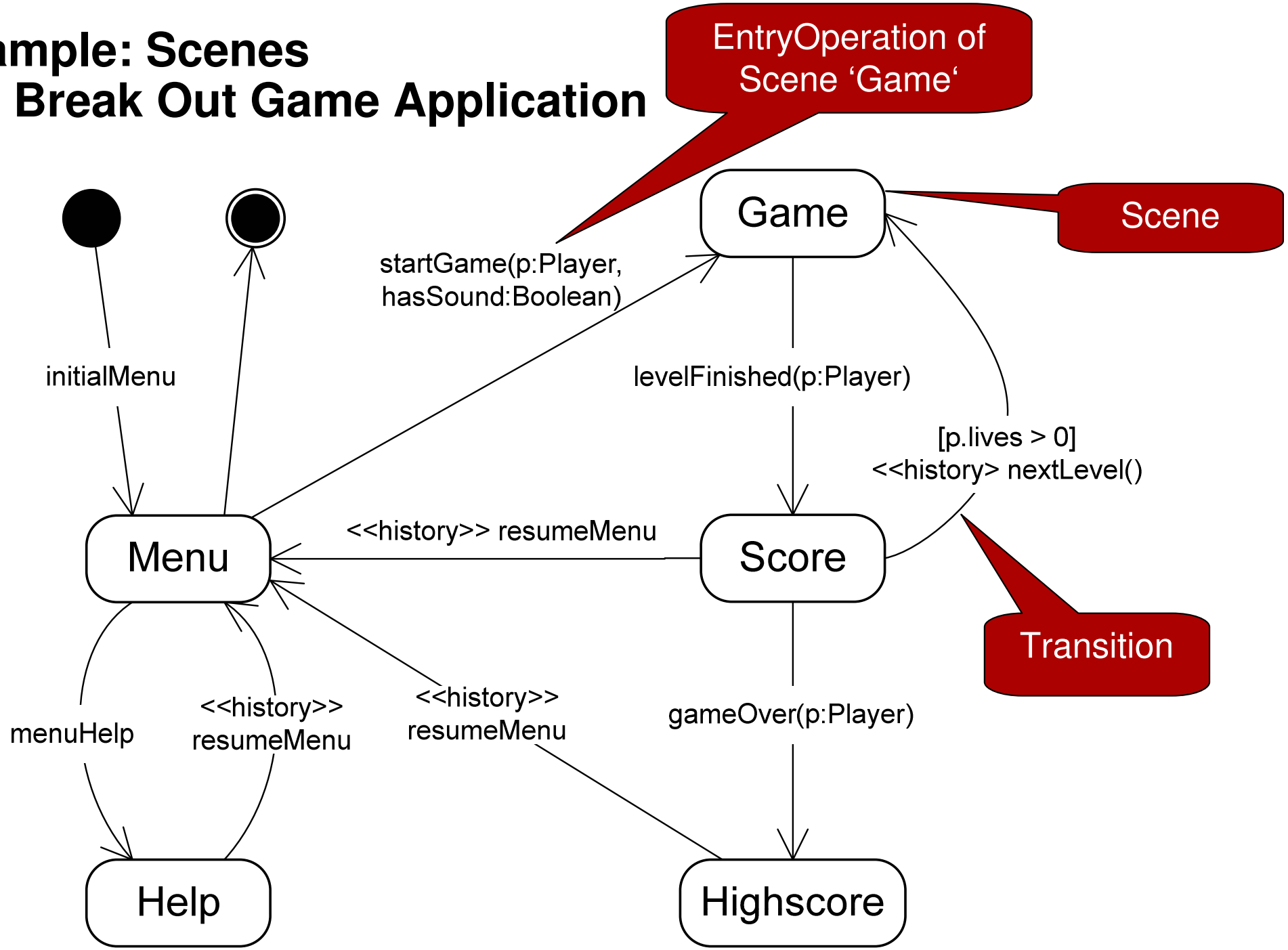
Notation:

- Adapted UML state charts

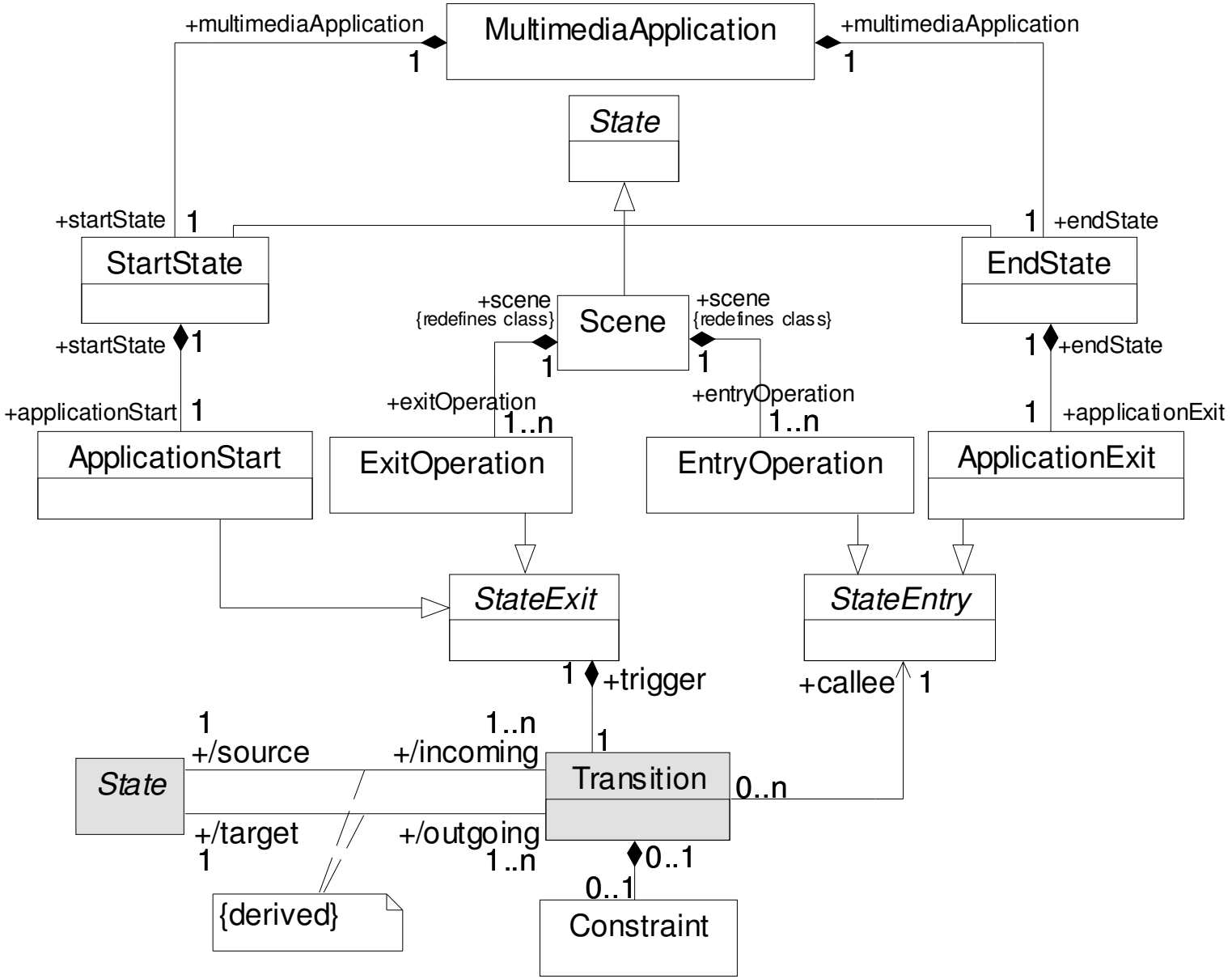
Specific model elements:

- *Scene:* represents a specific state of the user interface (e.g. a ‚screen‘)
  - Can have an internal state, i.e. class properties
- *EntryOperations*, *ExitOperations*: operations of a scene which are executed when scene is entered/exited
- *History*: Entry into a scene might sometimes require to resume the last state of the scene. Example: the user views a video, leaves scene to view the help, and wants to continue the video afterwards. The Keyword *history* specifies that a entryOperation of a scene resumes the scene‘s previous state.
- *Transitions* between scenes correspond to execution of exitOperation in the source scene and entryOperation in the target scene.
  - Name of addressed entryOperation is denoted next to the transition
- Scenes can have attributes and operations => additionally modeled as classes in the class diagram tagged with the keyword *scene*.

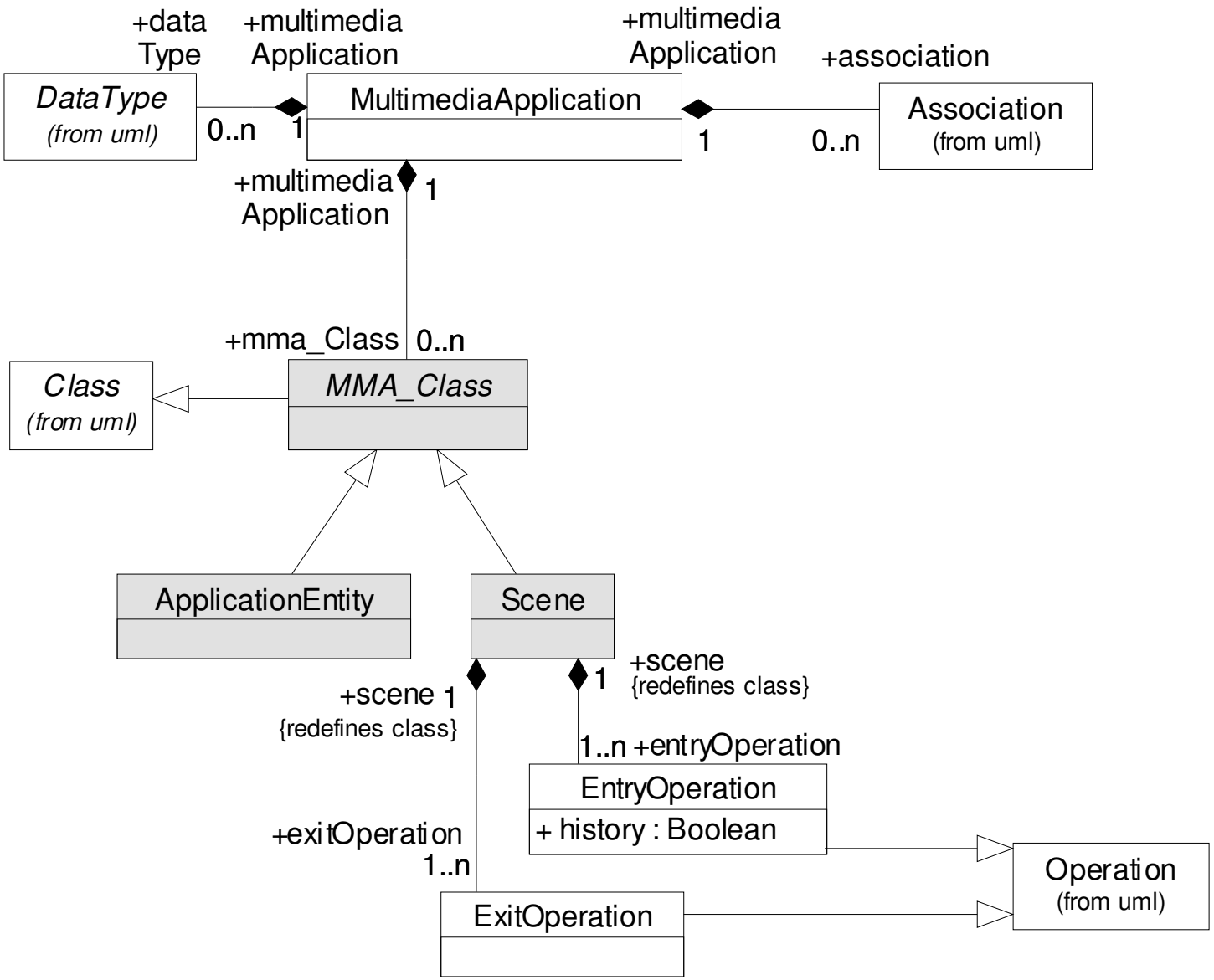
Example: Scenes for Break Out Game Application



Metamodel for scene diagram



Metamodel for MML-specific classes



Abstract User Interface Diagram

Motivation:

- Platform independent specification of a scene‘s user interface in terms of abstract user interface components
- Captures interaction elements resulting from the tasks to be fulfilled from the user or from user interface sketches
- User interface elements are related to elements from the application structure diagram

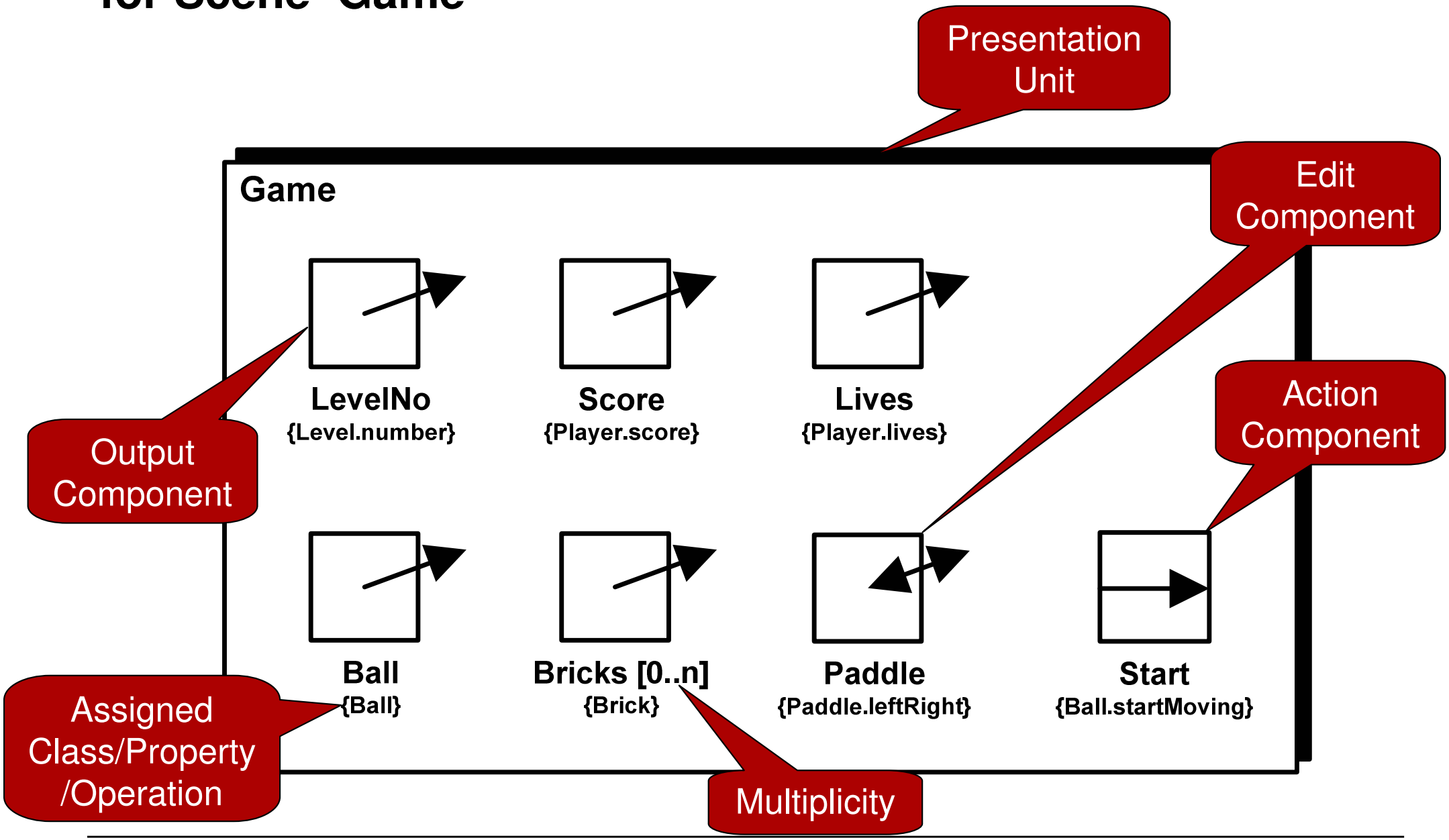
Notation:

- Similar to various user interface modeling languages (no corresponding UML diagram type)
- Can be combined with additional diagrams (e.g. concrete presentation diagram) or sketches to document a corresponding specific idea of the concrete layout

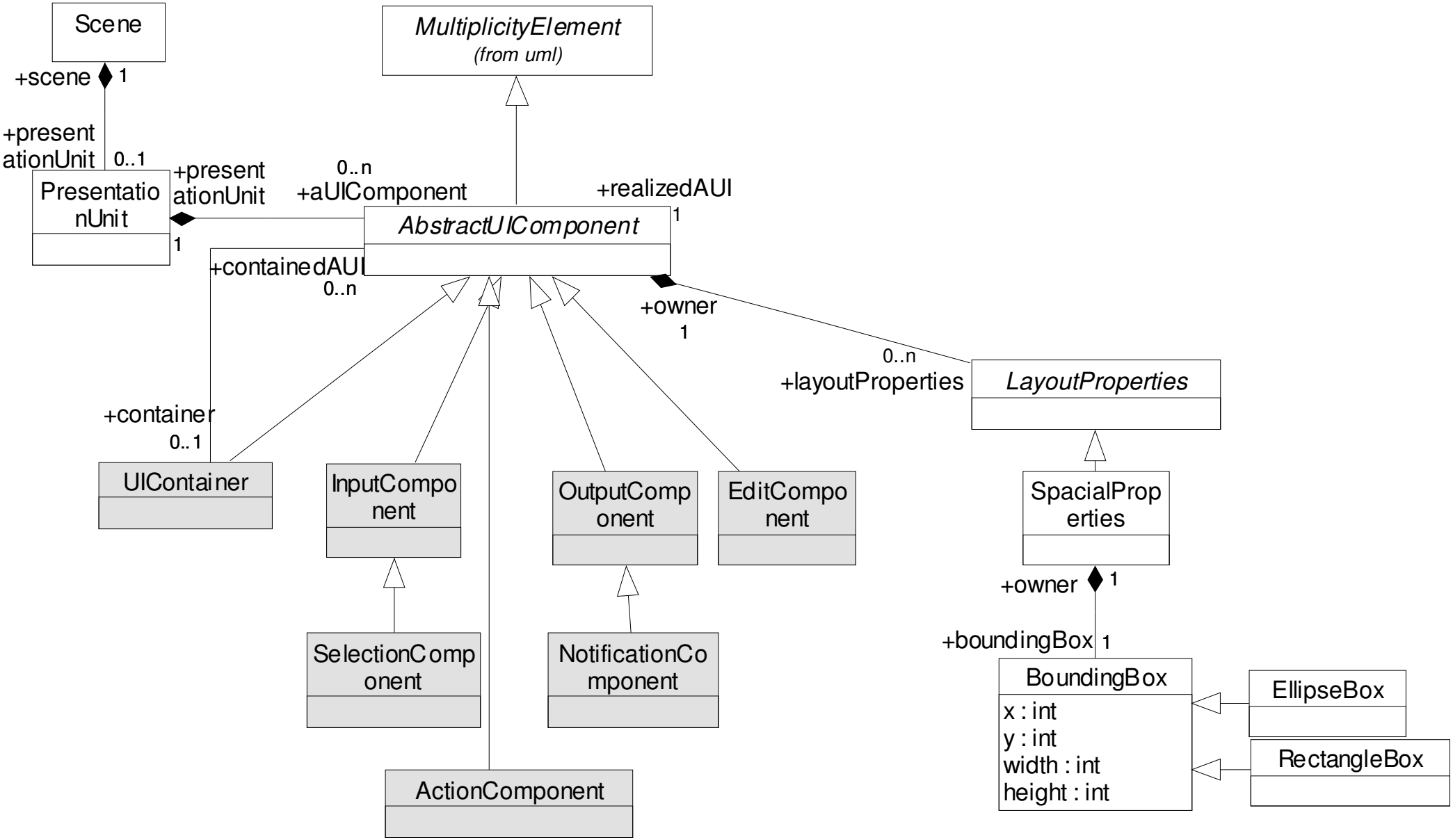
Specific model elements:

- *Abstract user interface components* (AUIs):
  - *Input Component*: allows the user to input some data; *Output Component*: presents data to the user; *Edit Component*: Combination of input and output component; *Action Component*: allows the user to invoke an action (without data input)
  - Multiplicity to specify multiple occurrence of an AUI
- *UIRepresentation*: relationship which specifies that an AUI is associated with a class (application entity or scene), class property, or class operation from the application structure diagram
  - Multiplicities to specify relationships other than 1:1
- *Presentation Unit:* Top-most container for the AUIs; assigned to a scene
- *UI container*: to be used for further structuring of the AUIs

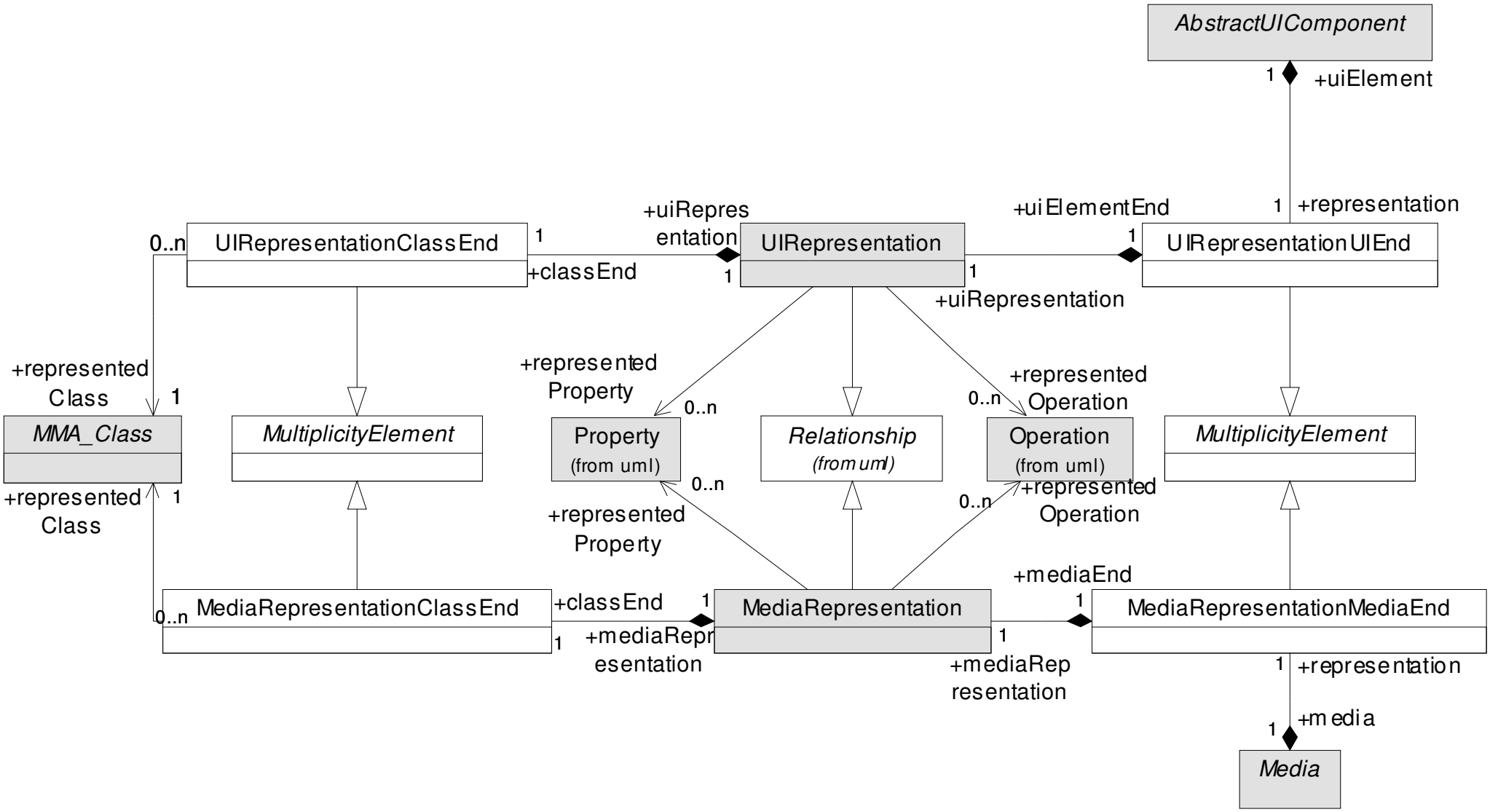
Example: Abstract User Interface for Scene ‘Game’



Metamodel for abstract user interface components



Metamodel for UIRepresentation and MediaRepresentation



Media User Interface

Motivation:

- Integration of media and user interface elements
- Some of the elements found during user interface design are realized by the application‘s media components, others are realized in no specific way (they can be realized by common standard widgets for the respective taget platform)
- The integration of temporal media components can lead to additional events not triggered from the user (e.g. collision of animations or end of a video)

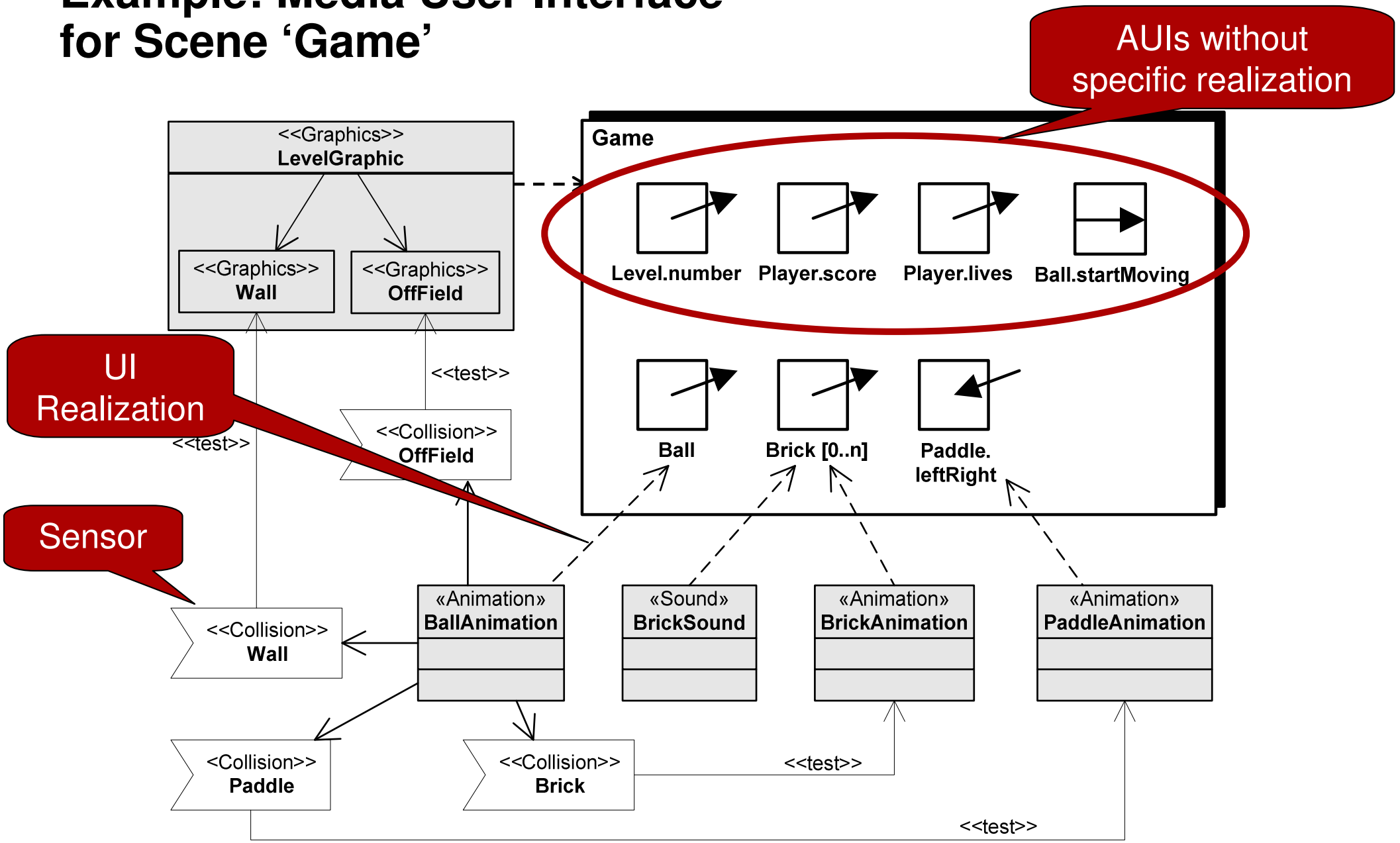
Notation:

- Abstract user interface diagram extended with realitionships to media components and with sensors

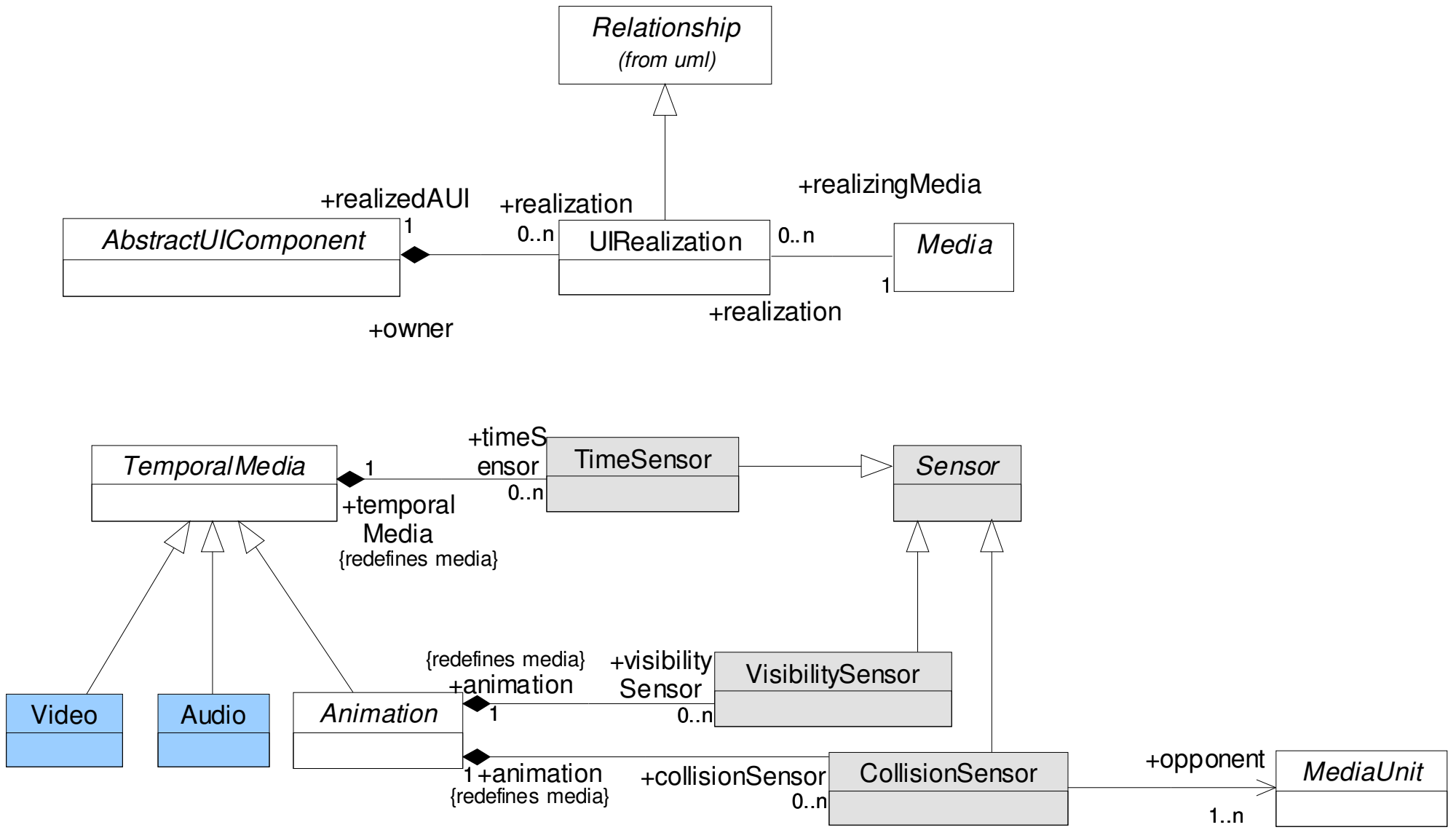
Specific model elements:

- *UIRealization*: Relationship which specifies that an AUI is realized by a media component from the application structure diagram
- *Sensor*: assigned to a media component to specify that the media component can trigger additional events

Example: Media User Interface for Scene 'Game'



Metamodel for UIRealization and sensors



Interaction Diagram

Motivation

- Overall interaction flow/dialogue between user and application
- Integrates events from AUIs/media components with the application logic

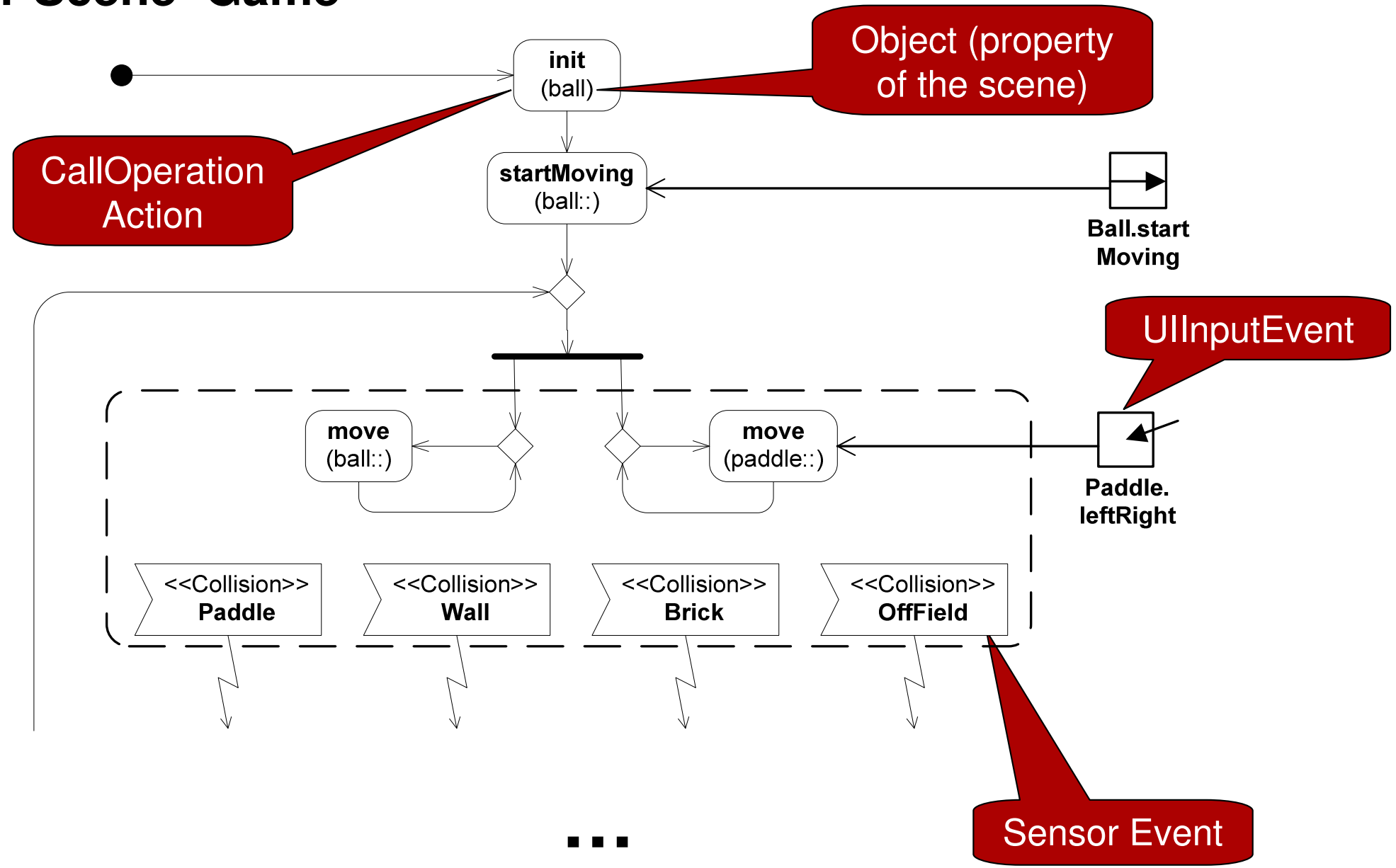
Notation:

- UML activity diagram with limited set of actions and with references to AUIs and sensors from the media user interface

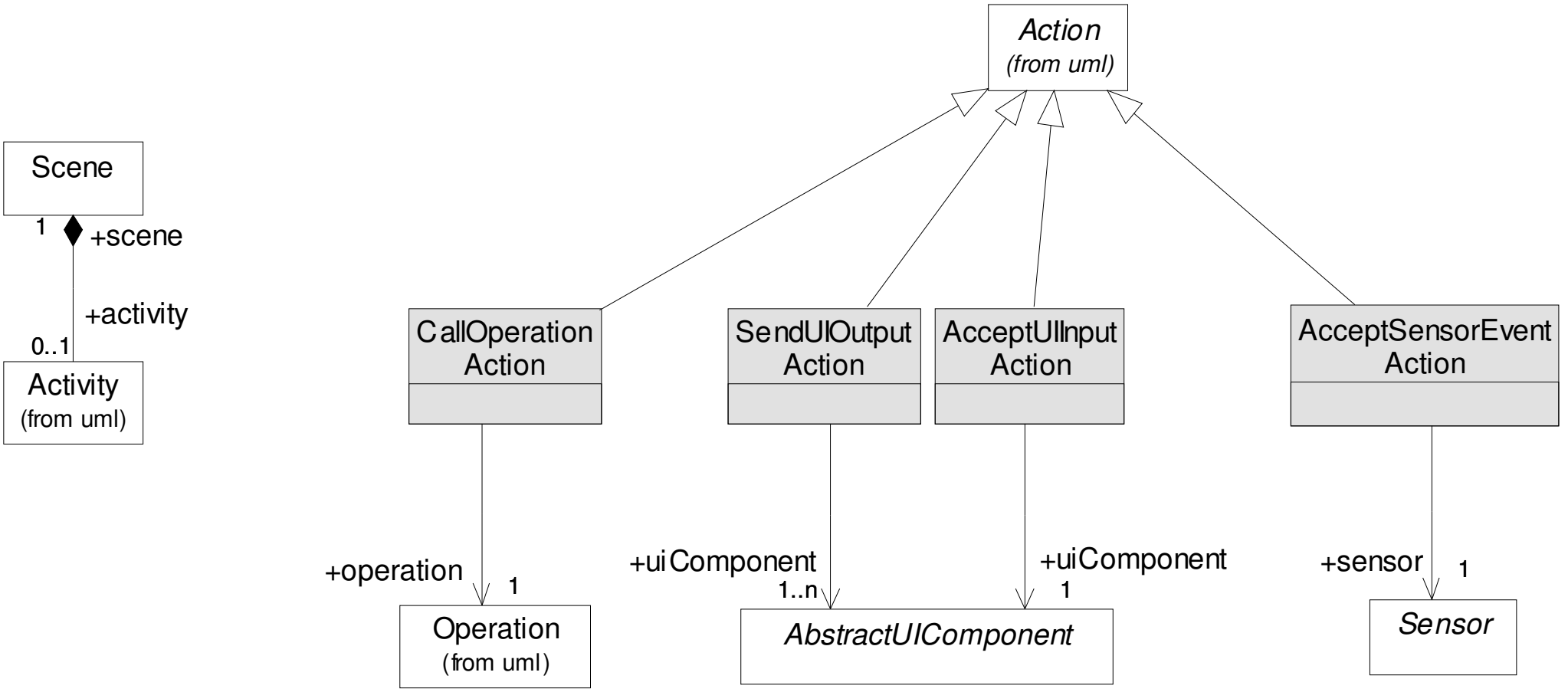
Specific model elements:

- *CallOperationActions*: calls an operation from the application structure diagram
- *SendUIOutputAction*: sends an output to an AUI from the media user interface diagram
- *AcceptUIInputAction*: receives an input from an AUI from the media user interface diagram
- *AcceptsSensorEventAction*: receives an event from a sensor from the media user interface
- To allow automatic code generation, no other kinds of (UML) actions are allowed to be used in the diagram

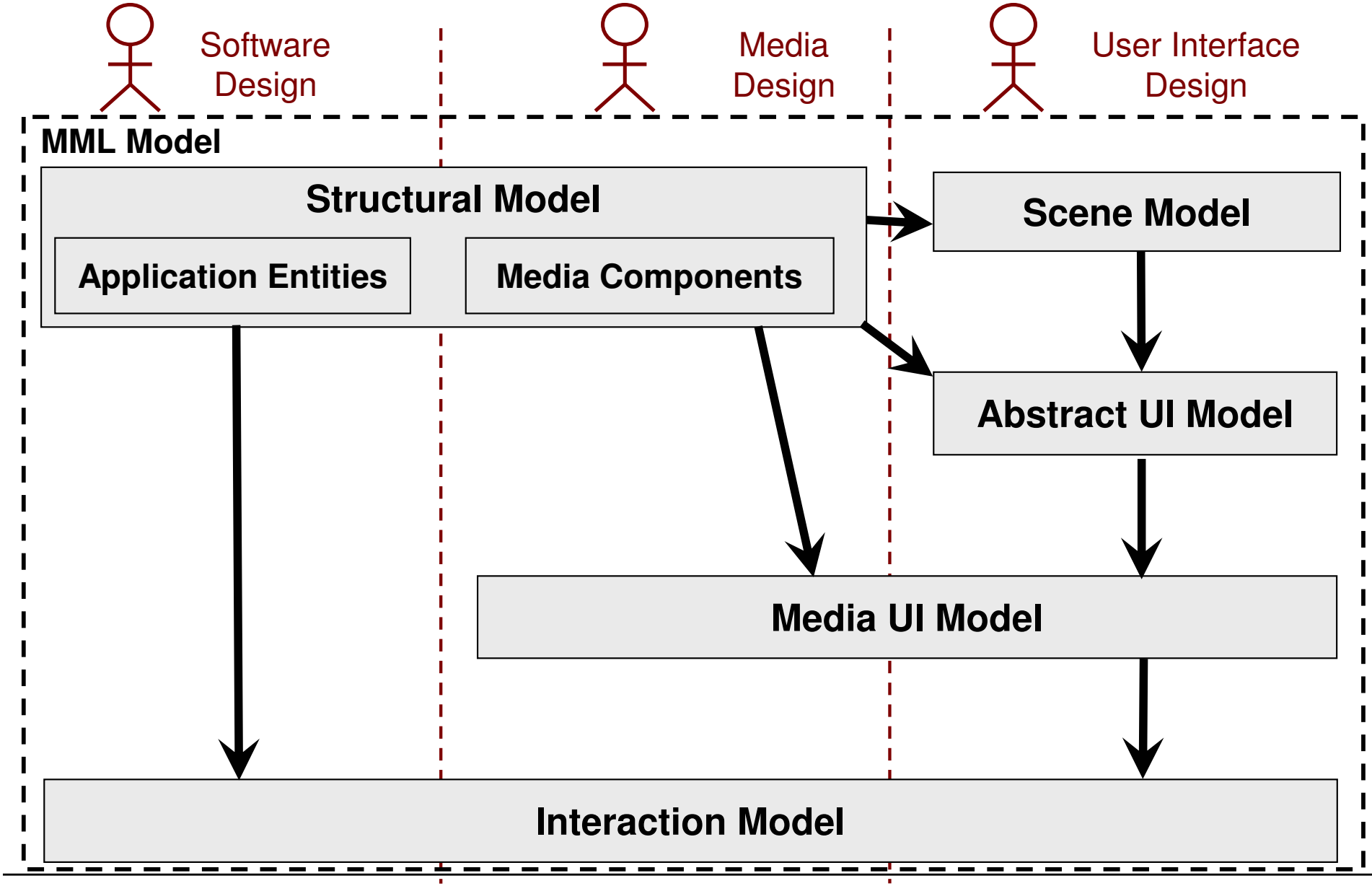
Example: Interaction diagram for Scene ‘Game’



Metamodel for MML-specific elements

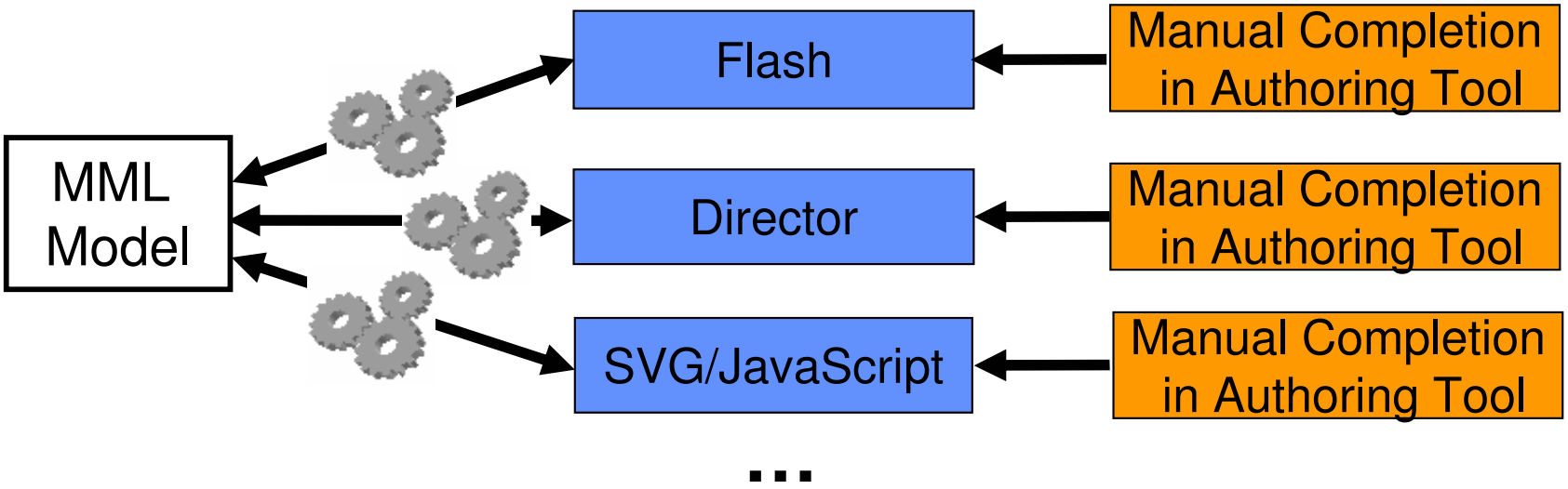


Design process:
Integration of the different developer groups



Code Generation: Integration of authoring tools

- How to integrate – for the creative design tasks - the powerful multimedia authoring tools into the model-driven development process?



Generate *code* for:

- Classes and class attributes
- Overall behavior
- Integration of media objects and the user interface

Generate *placeholders* for:

- Class operations
- Media objects
- User interface objects and layout

Structure and integration managed in model

Creative design performed in authoring tools

Details of Transformation into Code-Skeletons

Model	Generated Code
Multimedia Application	• FLA-File • ActionScript Class which loads the single scenes according to the scene diagram
Classes	ActionScript Classes (‘Model’, ‘Observable’)
Class Operations	Placeholders for operation bodies
Scenes	• FLA-File showing the scene‘s user interface, • ActionScript Class (‘Controller‘): entryOperations, exitOperations, code for interaction
Media Components	• FLA-File containing placeholders for all media components in its library; library will be used as shared library for the different scenes • ActionScript Class (‘View’, ‘Observer’)
Abstract UI Components	• Placeholders on the stage in the related scene; if a media component realizes the AUI, then the media component (from the library) is placed on the stage • ActionScript Class (‘View’, ‘Observer’)

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
- Problem: Next iteration of code generation can overwrite manual completions (of placeholders) in the authoring tool
- 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 a single 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