10 Modelling Multimedia Applications

Literature:

M. Jeckle, C. Rupp, J. Hahn, B. Zengler, S. Queins: UML Glasklar, Hanser Wissenschaft Muenchen, 2003

David Frankel: Model Driven Architecture, OMG Press, 2003

Models

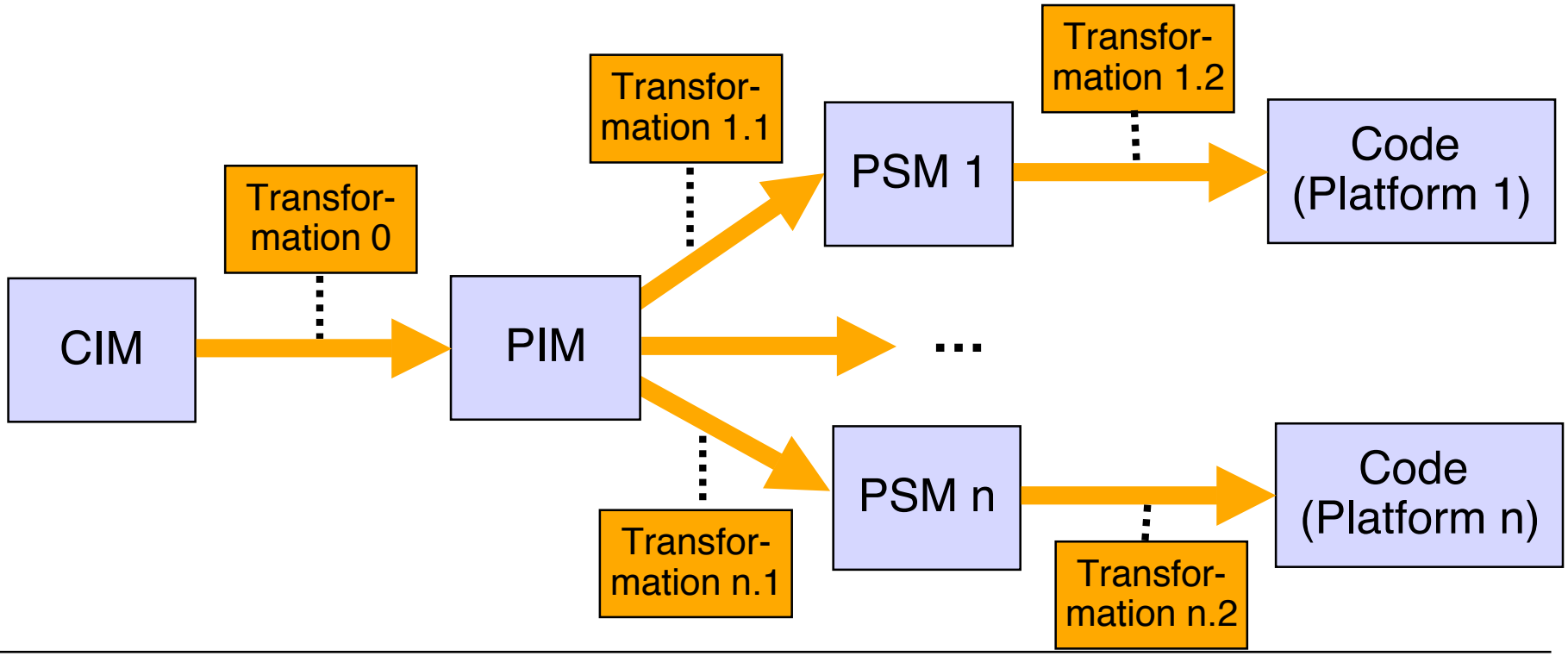
- How to denote the concepts found during analysis and design phase of the software development process?
  a) Non-formal: e.g. in natural language.
     – Pro: easy to read and write
     – Contra: large descriptions, often inconsistent and ambiguous
  b) Formal: e.g. with mathematical formulas
     – 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

Model-Driven Development

- Development process with models as core assets
- Idea:
  - ‘Programming’ on abstract conceptual level
  - Implementation code is generated automatically from models
  - Expert knowledge about implementation details is put into the code generator
- 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
    - » Backwards, to allow iterations
- Transformations specified explicitly and treated as assets of their own
  - Customizable

Model-Driven Architecture

- *Model-Driven Architecture* (MDA): A concrete framework defined by the *Object Management Group* (OMG) for model-driven development
  - CIM: Computation independent model
  - PIM: Platform independent model
  - PSM: Platform specific model



10 Modelling Multimedia Applications

10.1 Model-Driven Development

10.2 Multimedia Modeling Language MML 

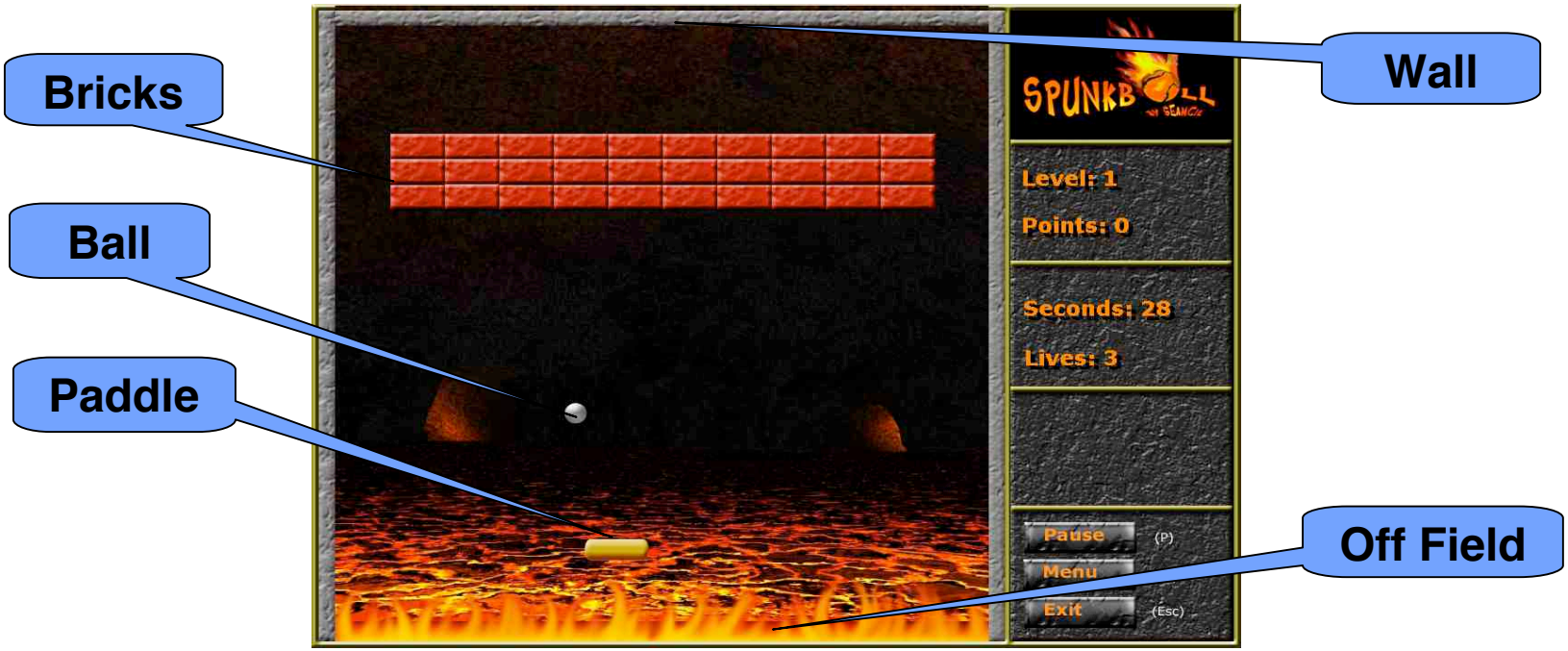
Literature:

A. Pleuß: MML - A Language for Modeling Interactive Multimedia Applications.
Seventh IEEE International Symposium on Multimedia (ISM 2005), pp. 465 - 473, IEEE Society Press, 2005

A Modeling Language for Multimedia Applications

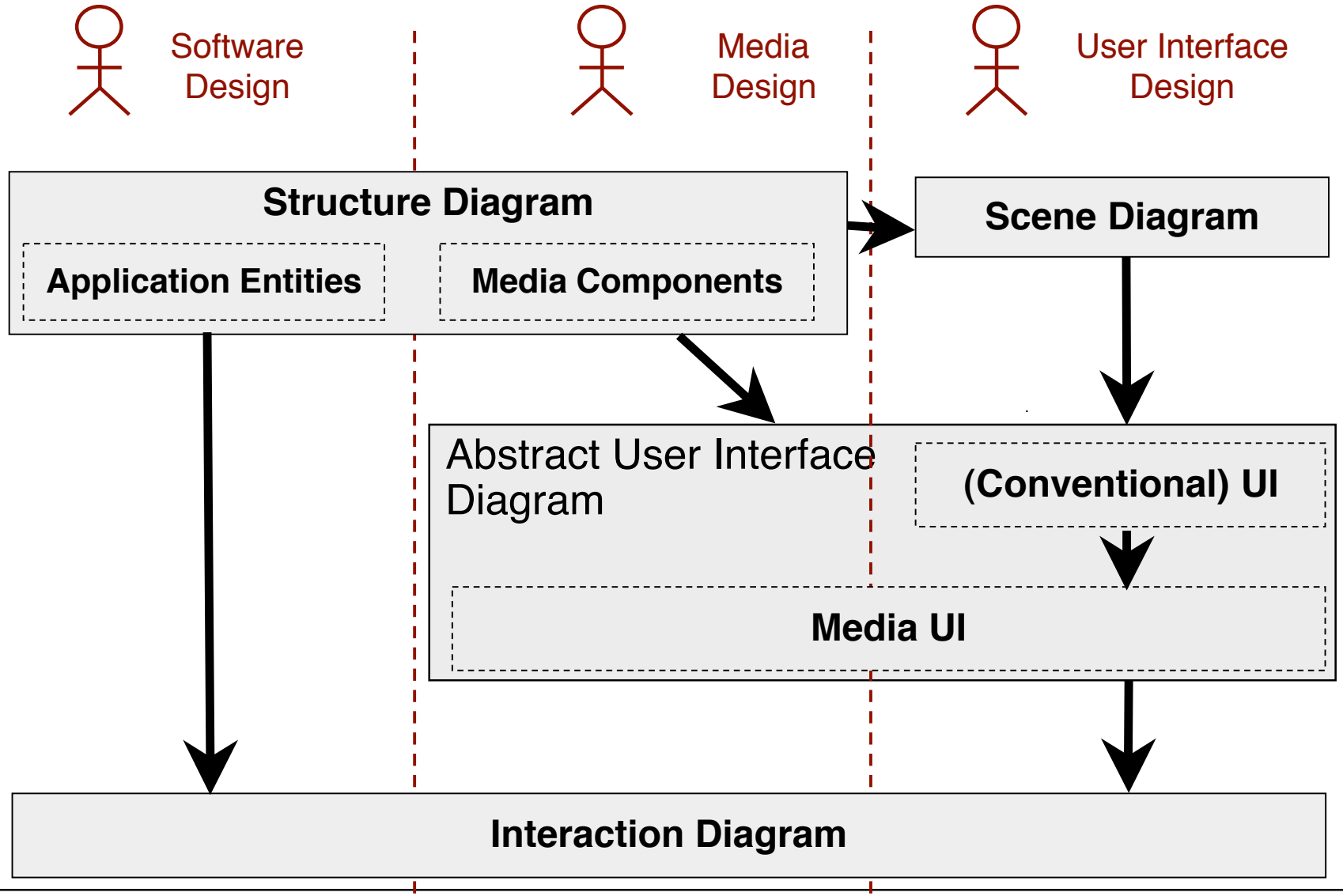
- *Multimedia Modeling Language* (*MML*) is a research approach (from LMU)
- MML is an independent modeling language which reuses concepts from UML (Unified Modeling Language)
- Problem: Tool support
  - Solution: Provided as a Profile („plug-in“) for UML
    - » MML available as Profile for the UML tool *MagicDraw*
- Most important element in a UML Profile: *Stereotype*
  - Extends or adapts an existing UML model element for a specific purpose
  - Example: UML Profile for Java contains a Stereotype <<JavaClass>>
  - Stereotypes denoted in guillemets «»

Example Application: Break Out Game

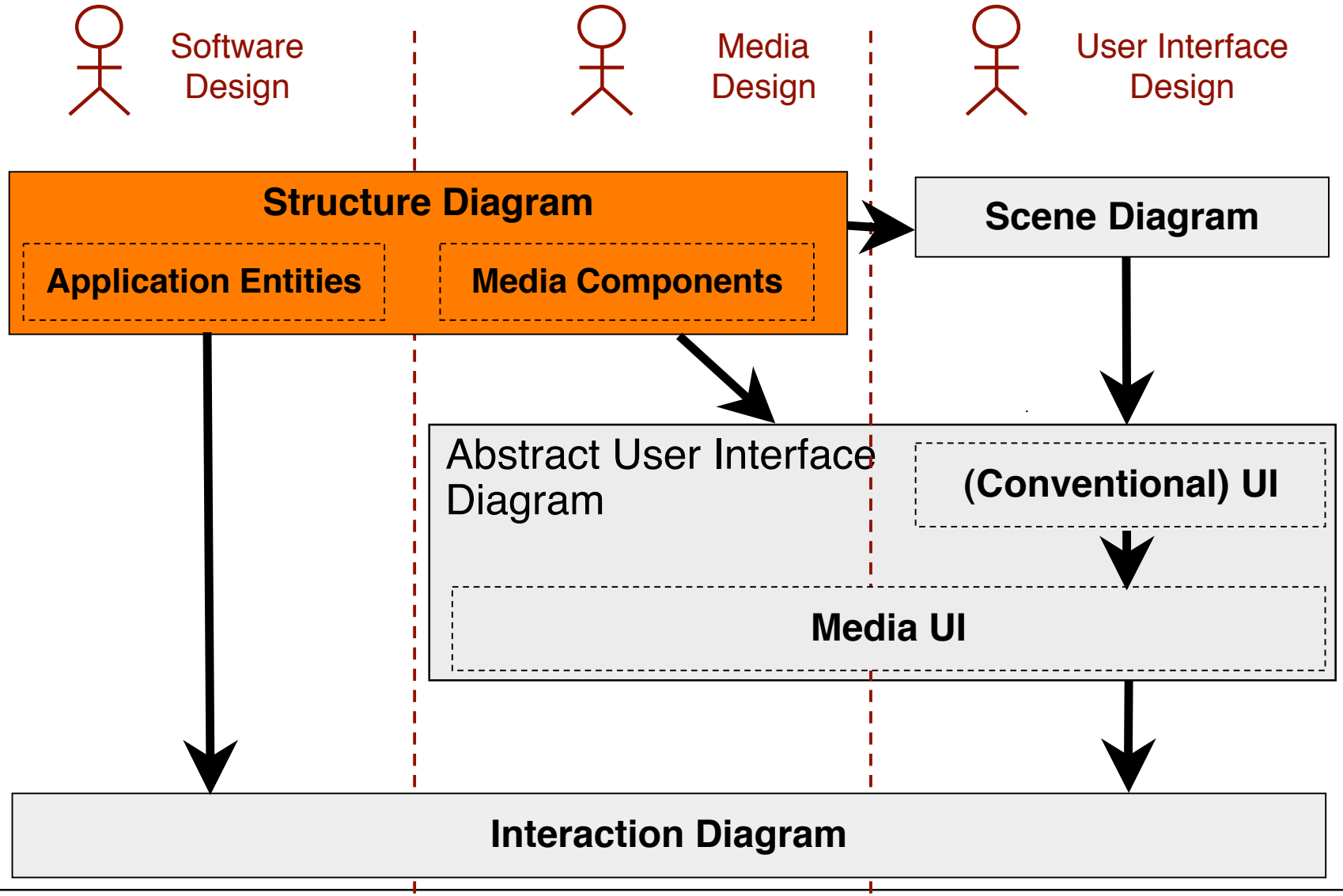


- (Small) games are good examples for interactive multimedia applications
  - Make intensive use of media objects, interaction and complex user interfaces
  - Functionality can be understood easily without specific domain knowledge

Diagrams in MML



Diagrams in MML



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

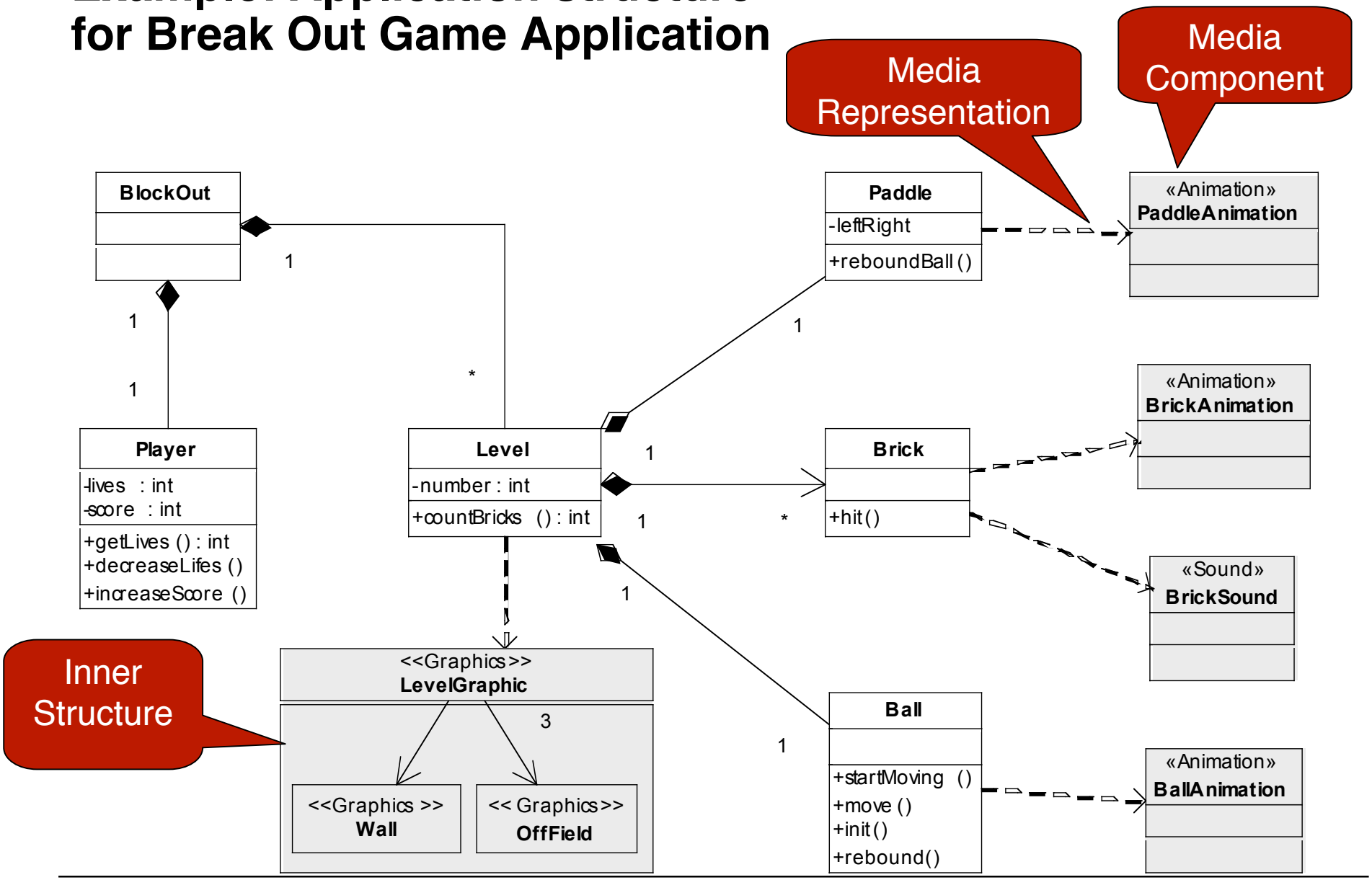
Notation:

- UML class diagram, extended with elements for media components

Parts:

- Application Entities for application logic
- Media components
- Scene classes (see Scene Model)

Example: Application structure for Break Out Game Application



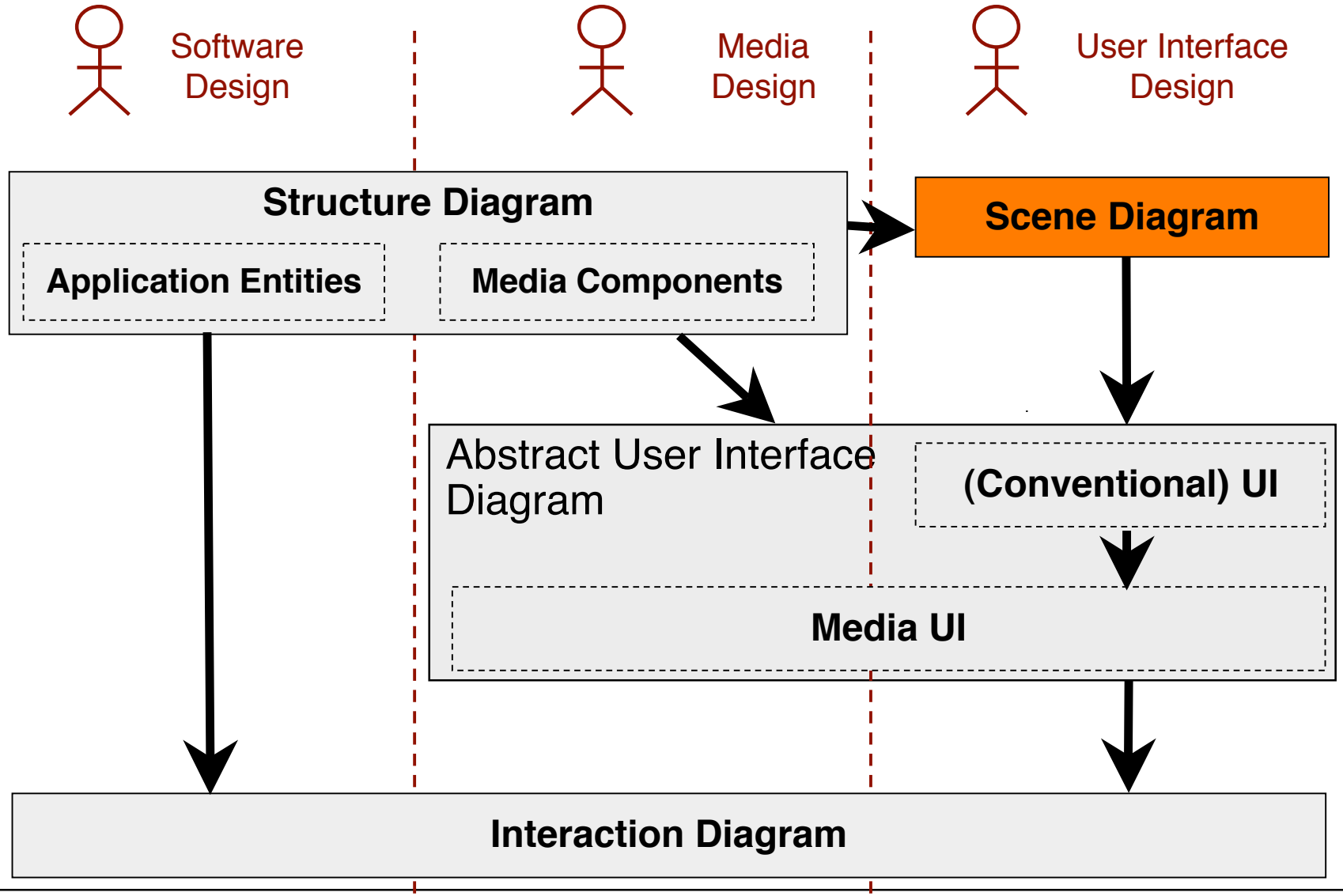
Application Structure: Application Entities

- Analogous to UML class diagram
- Classes for application logic marked as *application entities* to distinguish them from other kinds of classes
- Classes with attributes and operations
  - Attributes have a type and a default value
  - Operations may have parameters and a return value
- Associations between classes
  - Have a role name at each end
  - Have a multiplicity at each end, e.g. ‚1‘, ‚0..2‘ or ‚*‘ (default: 1)
  - Arrows show which ends are navigable (no arrows: bidirectional)
  - Aggregation or composition
- Generalizations between classes

Application Structure: Media Components

- *Media component* types: (*2D-)animation* (i.e. MovieClip in Flash), *3D-animation*, *audio*, *video*, *text*, *image*, *graphics*
- Media component includes (automatically) the standard functionality to present (and eventually manipulate) the media object
  - Image is decoded and displayed, video can be played, paused, stopped etc.
- Each media component represents an application entity
  - Specified in the model by *Media-Representation* relationship between application entity (class or part thereof) and media component
- Inner structure of media components can be specified if necessary
  - **Only** necessary if application logic must access inner parts
  - Inner components are connected with their parent by *Media-Composition* relationship
- Media Components can provide operations, e.g. *play()* for a video or *run()* and *jump()* for an animated character

Diagrams in MML



Scene Diagram

Motivation:

- Overall behavior and navigation
- Captures ideas e.g. from storyboards or derived from task models
- Shows the different “screens” of the application and the navigation between them
  - (however as MML is platform independent: a scene must not be necessarily be realized by a visual “screen”, for instance think of speech dialogue applications)

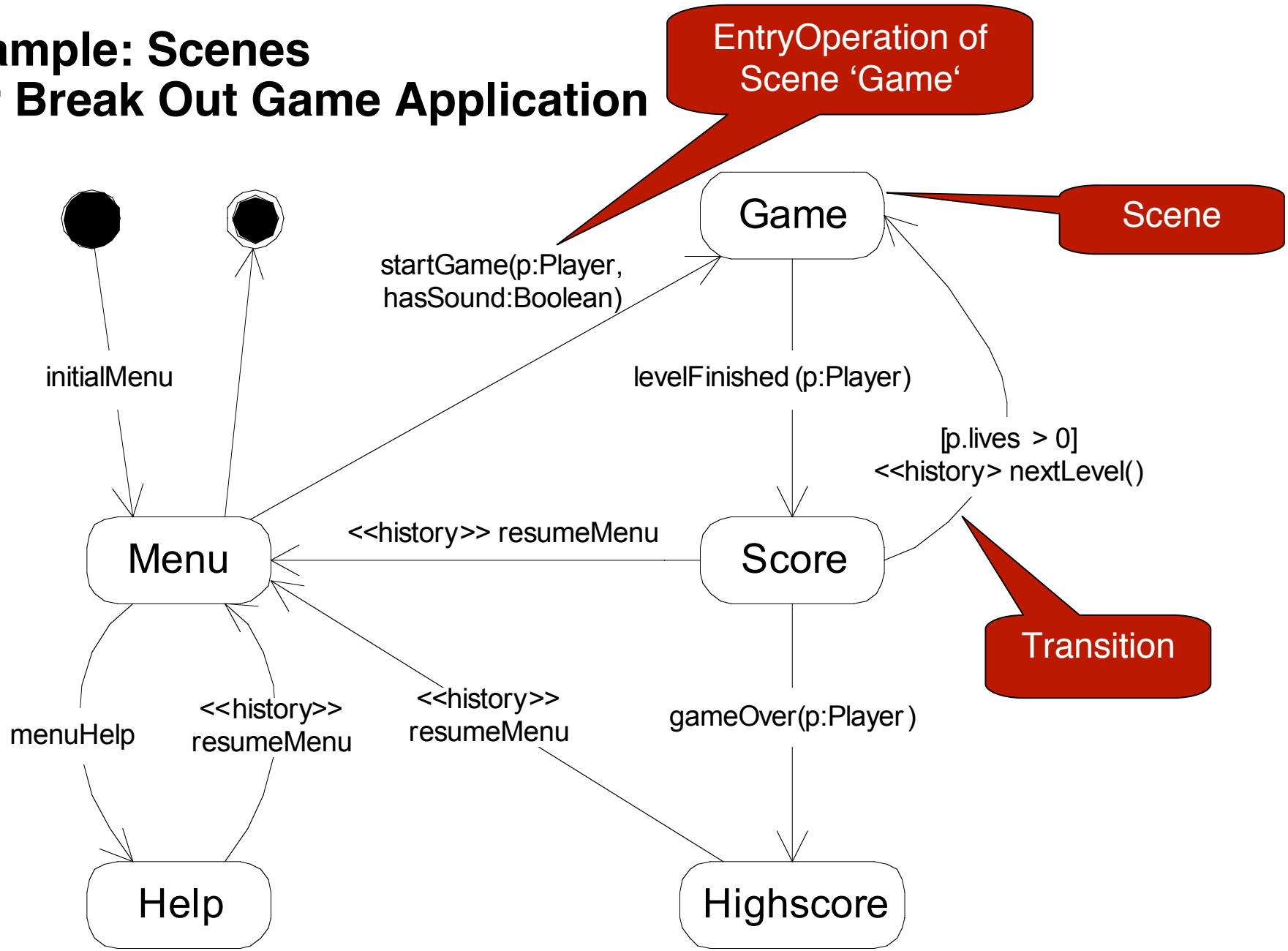
Parts:

- Scenes an Transitions between them

Notation:

- Adapted UML state charts

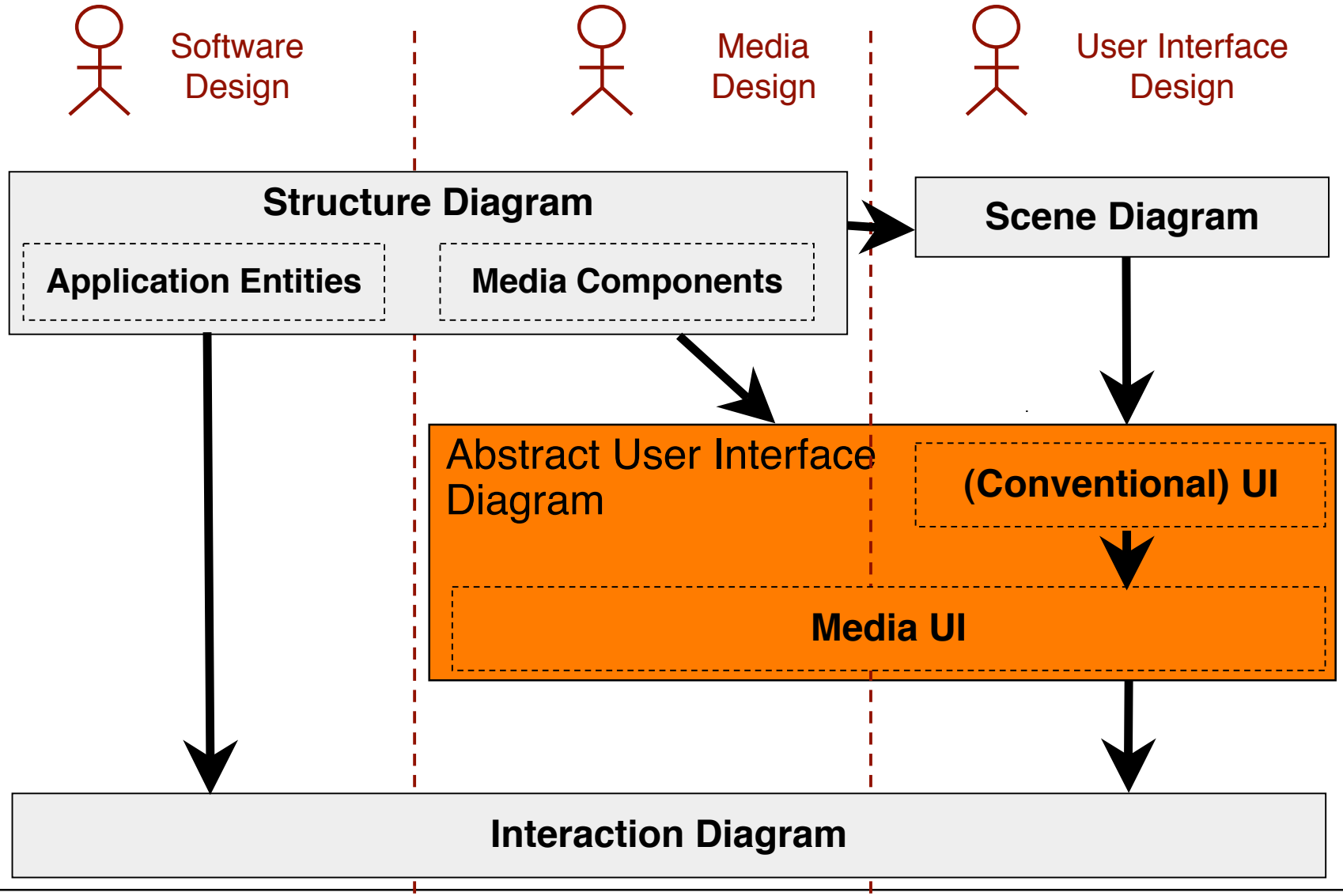
Example: Scenes for Break Out Game Application



Scenes in the Scene Diagram

- *Scene:* represents a specific state of the user interface (e.g. a ‚screen‘)
  - Can have an internal state, i.e. class properties
- *Entry-Operations*, *Exit-Operations*: specific kind of operations of a scene which are executed when scene is entered/exited
- *Transitions* between scenes correspond to execution of exit-operation in the source scene and entry-operation in the target scene.
  - Name of addressed entry-operation is denoted next to the transition
- *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.
  - Keyword *history* specifies that an entry-operation of a scene resumes the scene‘s previous state.
- Scenes can have attributes and operations => additionally modeled as classes in the class diagram tagged with the keyword *scene*.

Diagrams in MML



Abstract User Interface Diagram

Motivation:

- Platform independent specification of a scene's user interface
- Specifies the elements required to allow the user to fulfill all his task
- Derived e.g. from task analysis, storyboards, mock-ups, etc.
- User interface components represent application entities from the application structure diagram
- In a multimedia application, user interface components are partially realized by media components

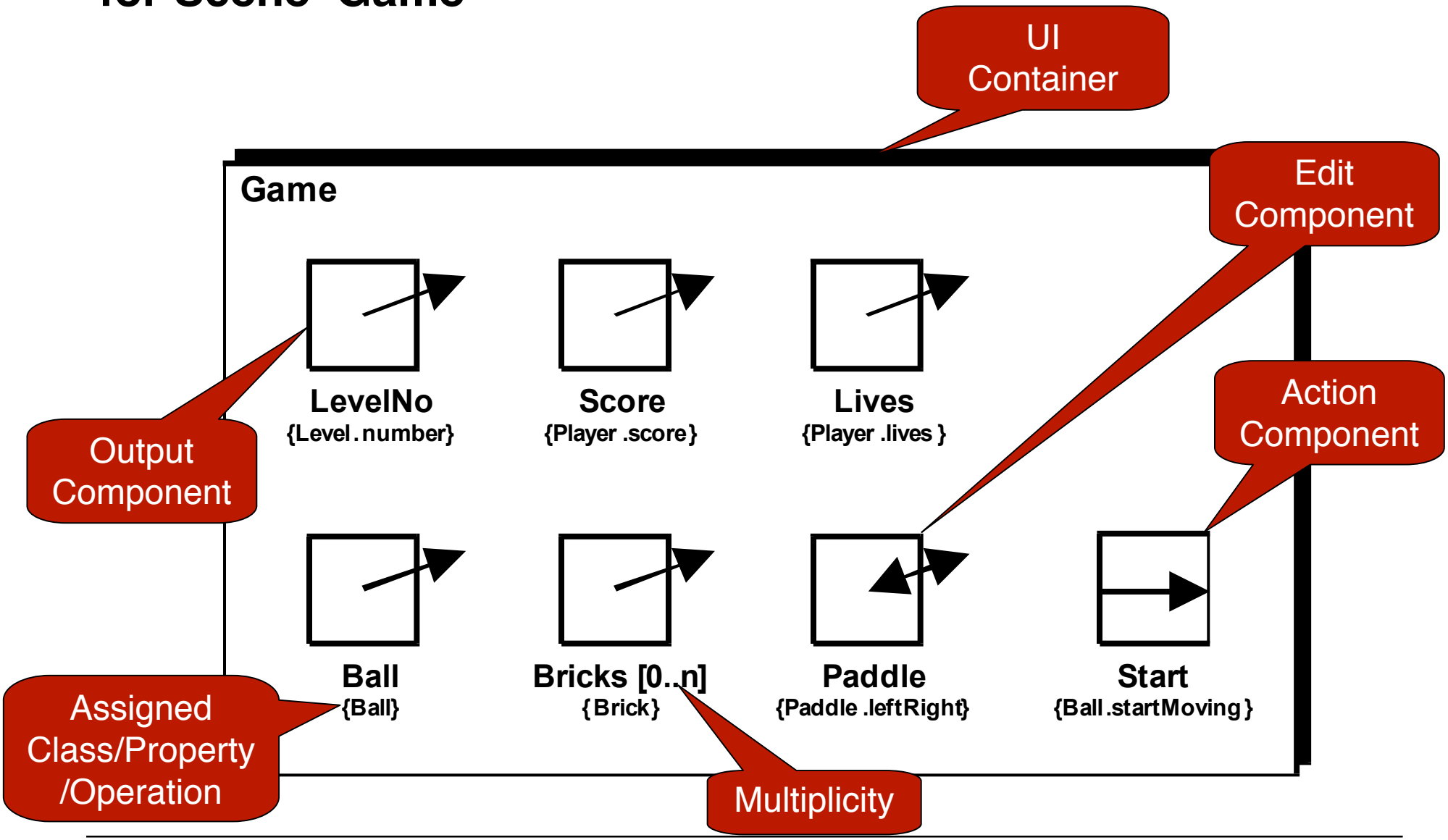
Notation:

- 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

Parts:

- For each scene:
  - User interface components and UI-Representations
  - Media Components and UI-Realizations
  - Sensors

Example: Abstract User Interface for Scene ‘Game’



Abstract User Interface: User Interface Components

- Each Scene has exactly one presentation unit containing the scene‘s user interface in terms of abstract User Interface Components
- Abstract User Interface Components (UIC):
  - *Input-Components*: Allows the User to input data (like a textfield)
  - *Output-Componen*t: Provides Information to the User (like a text label)
  - *Edit-Component*: Provides the User information and allows to edit it (like a textfield containing text)
  - *Action-Component*: Allows the User to invoke an Action without data input (like a button)
  - *Selection-Component*: Specialization of Edit-Component which allows the user to select from a set of items
  - *Notification-Component*: Specialization of Output-Component used to notify the user on specific situations (like a message box)
  - *UI-Container* used to structure UICs (like a Panel)
- UIC can have multiplicity to specify the number of its instances in the presentation unit (default: 1)

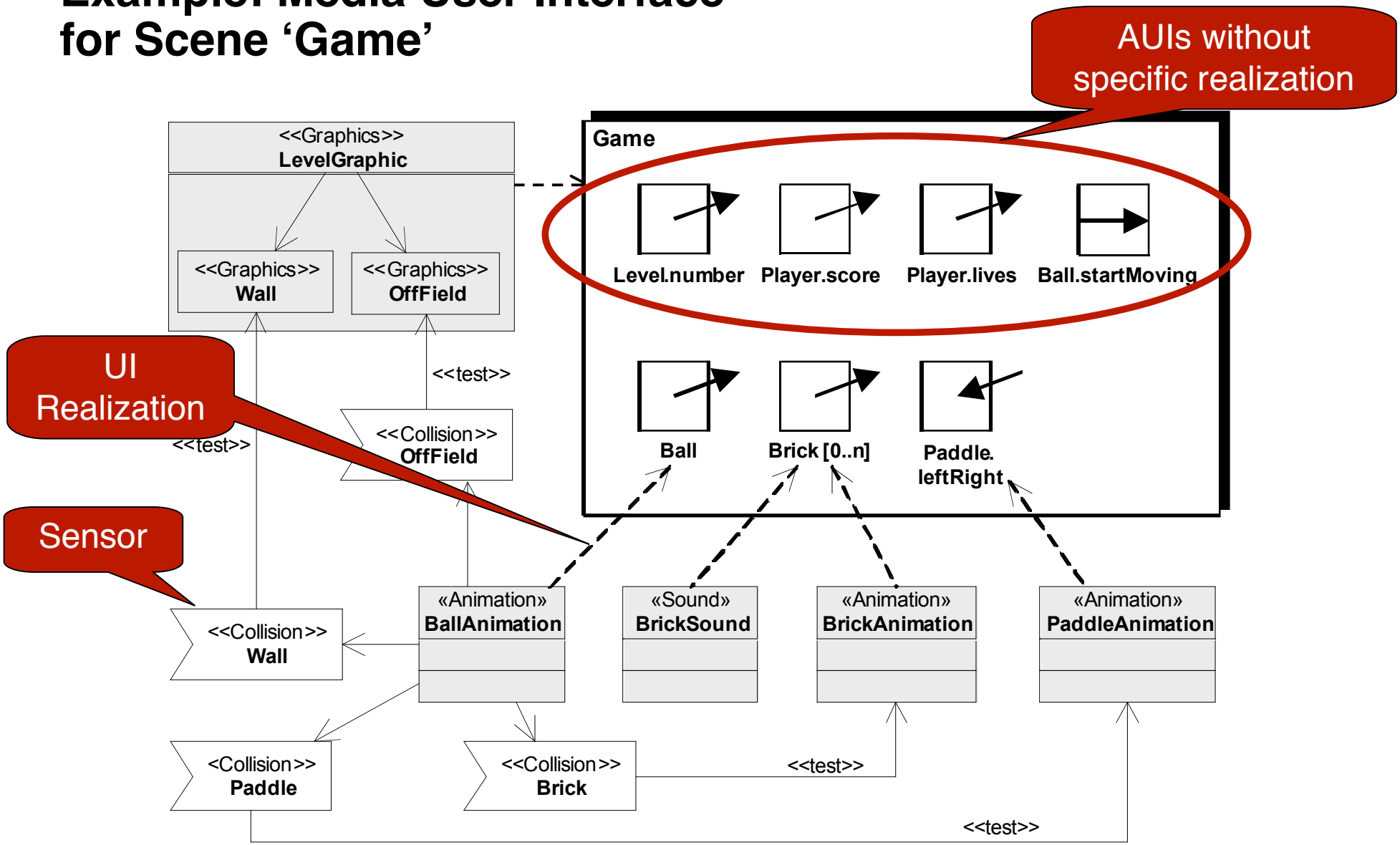
Abstract User Interface: UI-Realizations

- Media Components from application structure diagram can realize UICs
- Specified by UI-Realization relationship
- Consequence for implementation:
  - UICs realized by media components means that the media component is used on this user interface and (in addition) provides the functionality of the respective UIC (e.g. listens to mouse events)
  - Remaining UICs are implemented by conventional widgets (buttons, textfields, checkbox, etc.)

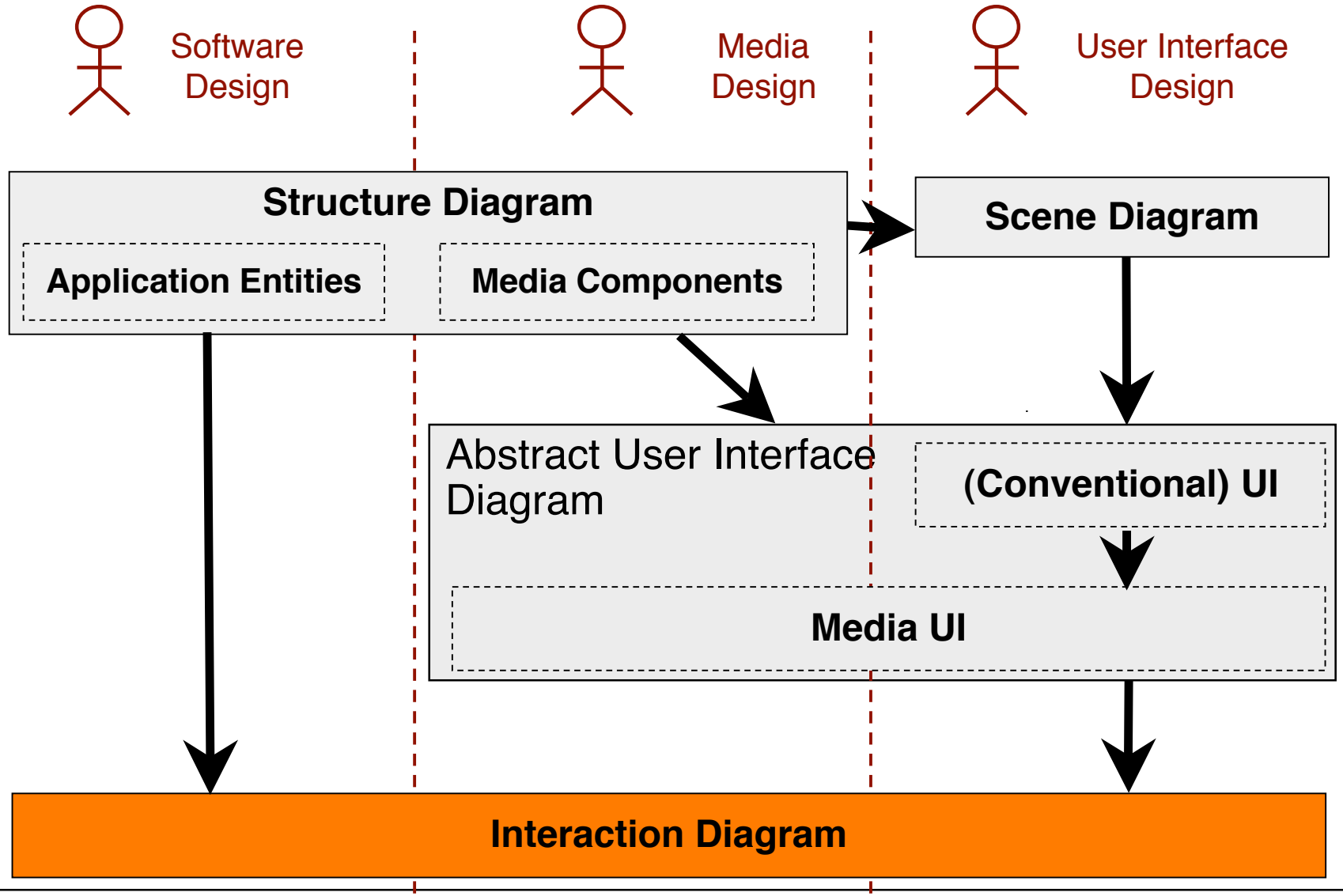
Abstract User Interface: Sensors

- Temporal media components may cause additional events (e.g. termination of a video, collision of an animation)
- *Time Sensor*: notifies about temporal events
  - End of a video
  - Specific time interval
- *Collision Sensor*: notifies about collision of animations with other media components
  - Relationship test specifies which other media components are observed
- *Visibility Sensor*: notifies if a media objects becomes visible/invisible (e.g. if a media object becomes masked by a moving animation)

Example: Media User Interface for Scene ‘Game’



Diagrams in MML



Interaction Diagram

Motivation

- Overall interaction flow/dialogue between the user and the application
- Integrates events from UICs and sensors with the application logic

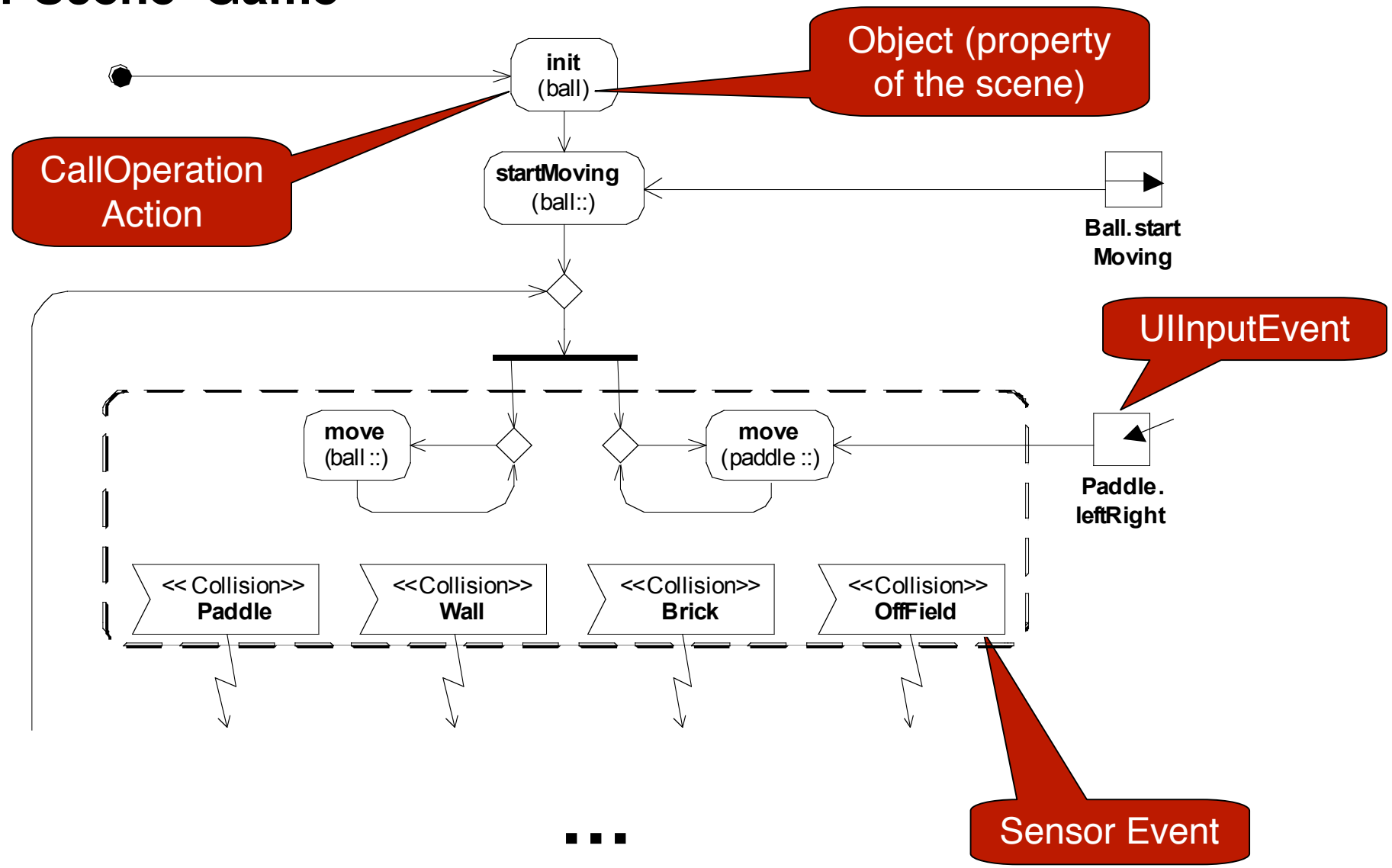
Parts:

- For each scene an activity

Notation:

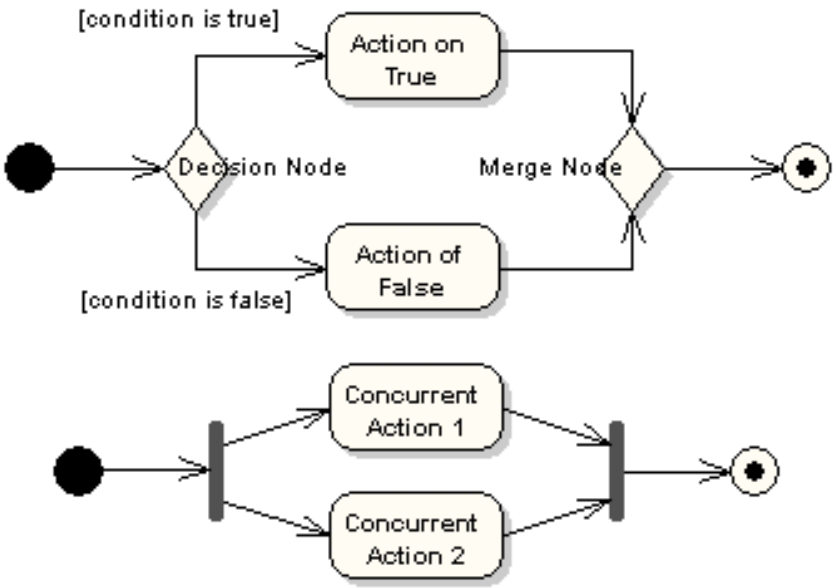
- UML activity diagram with limited set of actions and with references to AUIs and sensors from the media user interface

Example: Interaction diagram for Scene ‘Game’

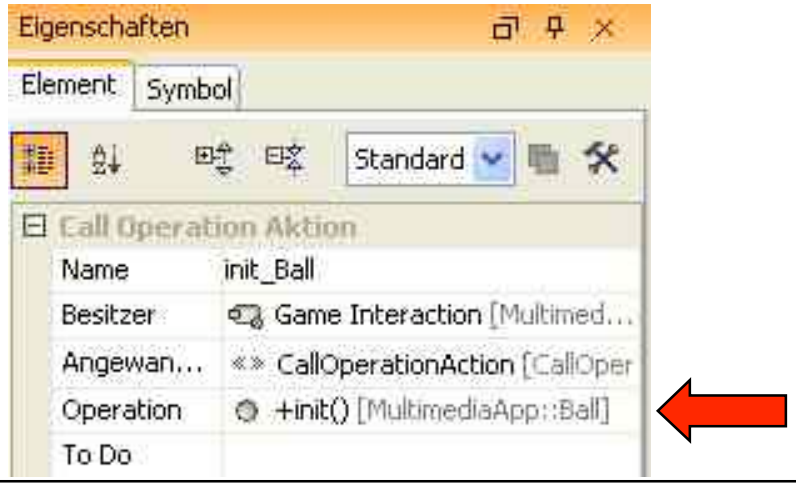


Interaction Diagram

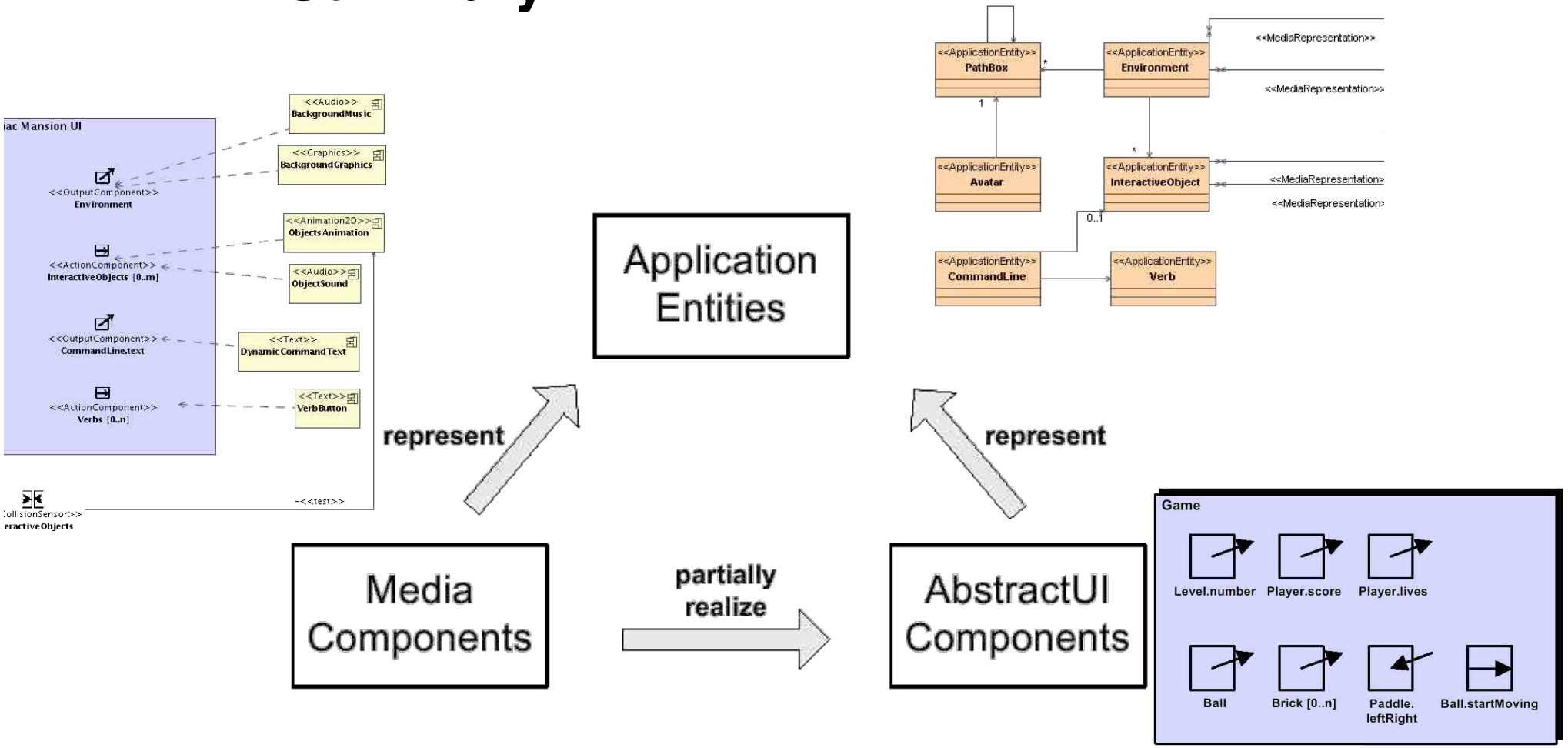
- Program flow is specified like in UML Activity Diagrams
  - Start Node and End Node
  - Decision Node and Merge Node for decisions („if“)
  - Fork Node and Join Node for parallel actions



- Action in MML: Calls an operation from the structural model (all Actions in MML are of type *CallOperationAction*)
- In MagicDraw: Target operation is specified in the Action's property window (see figure: operation *init()* from application entity *Ball*)



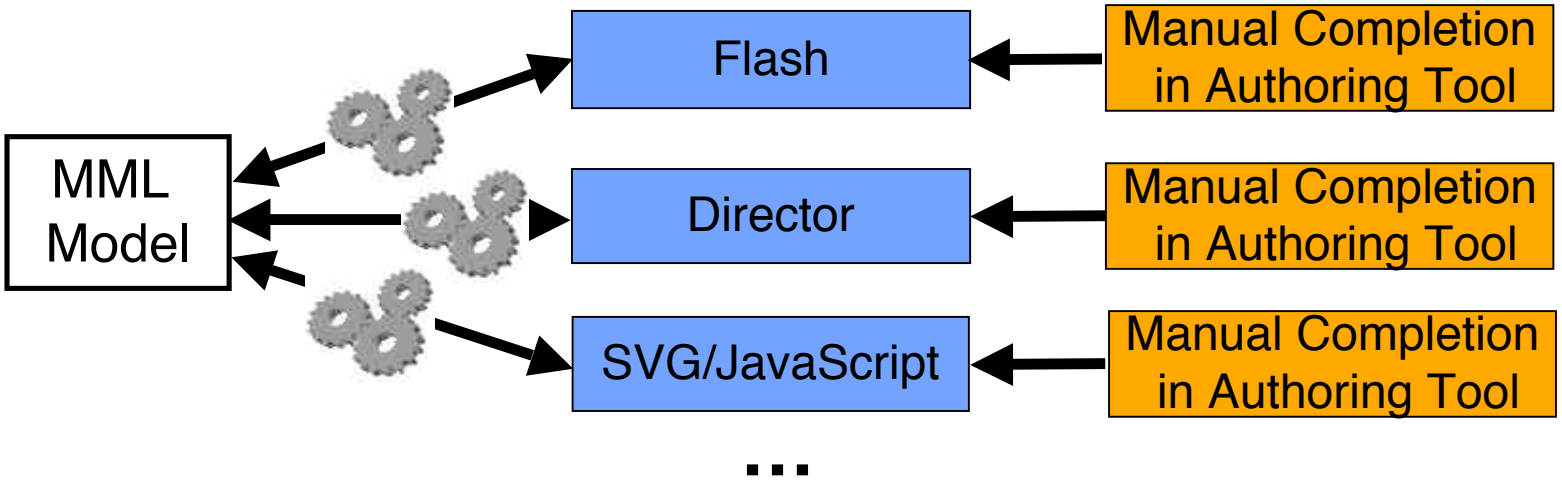
MML: Summary



+ Behavior:
- Scene Diagram for behavior between scenes
- Interaction Diagram for behavior within scenes

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

Transformation into Code Skeletons

Example: Code Generation for Flash/ActionScript (2.0)

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’)

Pros and Cons of Model-Driven Development for Multimedia Application

- Advantages:
  - Switch in platform (ideally) requires only change of code generation transformations
    - » E.g. from ActionScript 2 to ActionScript 3, from Flash to Silverlight
  - Higher level of abstraction leads to deeper analysis
  - Code generators can help to create well-structured code (e.g. modular Flash applications)
- Disadvantages:
  - Full code generation not (yet) possible, platform-specific completions prohibit easy switching between platforms
  - Round-trip engineering still needs to be developed
  - Writing abstract specifications is not attractive for multimedia developers
- Open issue:
  - What is the right language level for integrating the various design views/activities?