3 Development process for multimedia projects

Literature:

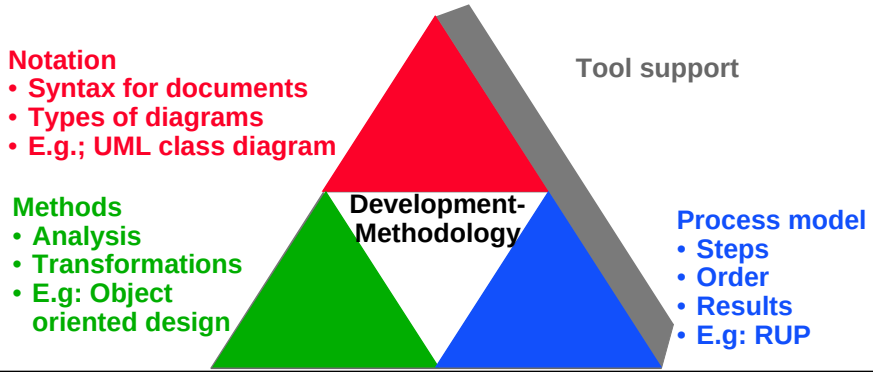
M. Jeckle, C. Rupp, J. Hahn, B. Zengler, S. Queins: UML Glasklar, Hanser Wissenschaft Muenchen, 2003

David Frankel: Model Driven Architecture, OMG Press, 2003

Outline

Development Process

- How to develope (large) interactive multimedia applications in a team?
- In Software Development: Development Methodologies
- Goals:
  - Remind on all required steps based on experience from previous projects
  - Structure process and allow scheduling of single steps
  - Enhance structure and quality of results



Concrete Example for Your Project Work

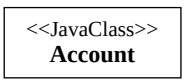
- Popular example (method & notation): Object-oriented design-phase using the Unified Modeling Language (UML)
- Basically, independent from selection of process model
- Advantages of visual modeling languages like UML:
  - More precise and compact than informal documents like text or pictures
  - Less complex and difficult to understand than formal methods like predicate logic
- In this lesson we discuss design-phase for multimedia applications using visual modeling languages
  - Should be used for your project
  - Provides an abstract overview (in terms of models) on the concepts shown in the previous chapters on the example technology Flash
- Discussion of alternatives and process models in the subsequent lessons

Are Software Engineering Concepts sufficient for Multimedia Application Development?

- Basically, an interactive multimedia application could be regarded as software
- Characteristics:
  - Integration of media objects like sound, video, animation, 3D graphics, etc.
  - High importance of the user interface
  - High degree of interaction
- Consequence: Different kinds of design in multimedia development
  - Software design
  - But also very important: User Interface Design
  - But also very important: Media Design (may require huge effort!)
- Plain UML not sufficient:
  - Low support for user interface elements
  - Barely support for media objects
- Required: Modeling language adapted to the characteristics of multimedia applications

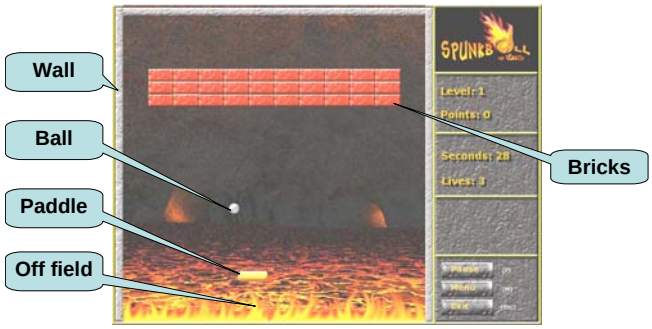
A Modeling Language for Multimedia Applications

- No standard available
- One research approach considered here: *Multimedia Modeling Language* (*MML*)
- MML is an independent modeling language which reuses concepts from UML
- Problem: Tool support
- Solution: Provided as a Profile („plug-in“) for UML
  - Many UML tools support UML Profiles
  - 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>> (in contrast to UML classes, Java classes do not support multiple inheritance)
  - Stereotypes denoted in guillemets «»



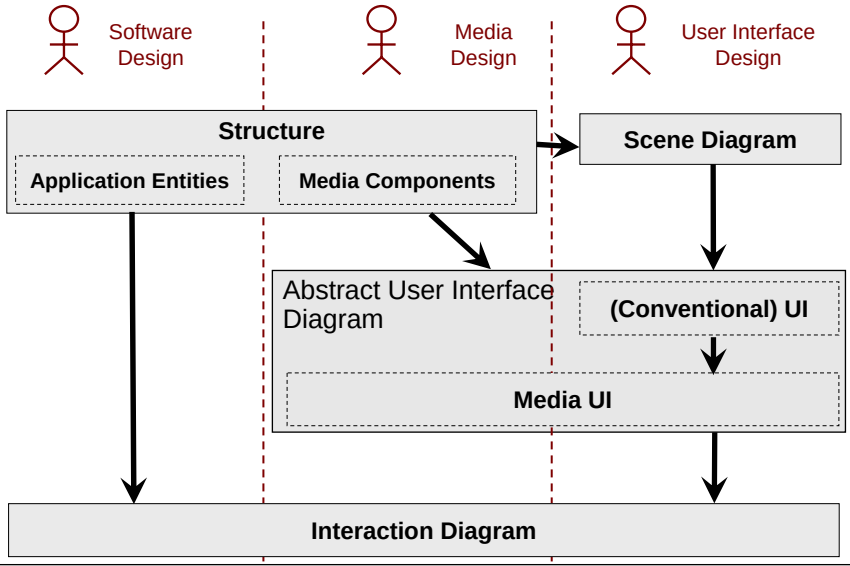
- Diagrams on the following slides are taken from MagicDraw (using MML-plugin)
  - For explanation purposes, all Stereotypes are visible in the diagrams
  - To get clearer diagrams it is also possible to hide Stereotypes in MagicDraw diagrams

Example Application: Break Out Game

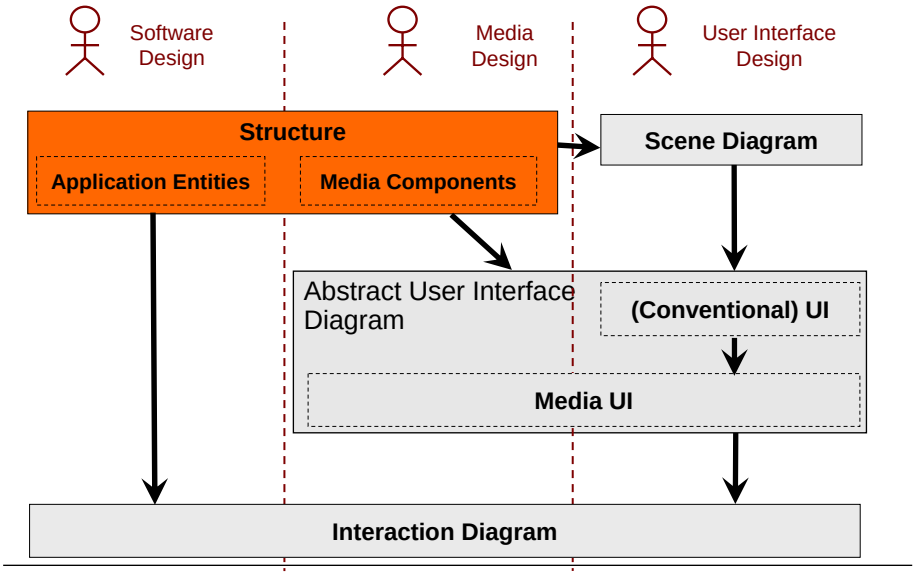


- (Smaller) games are good examples for interactive multimedia applications
  - They traditionally make intensive use of media objects, interaction and complex user interfaces
  - Their functionality can be easily understood without specific domain knowledge

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

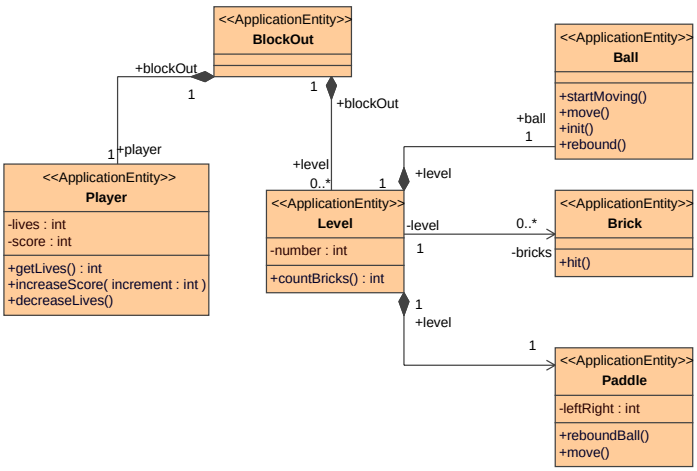
Parts:

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

Notation:

- UML class diagram, extended with elements for media components

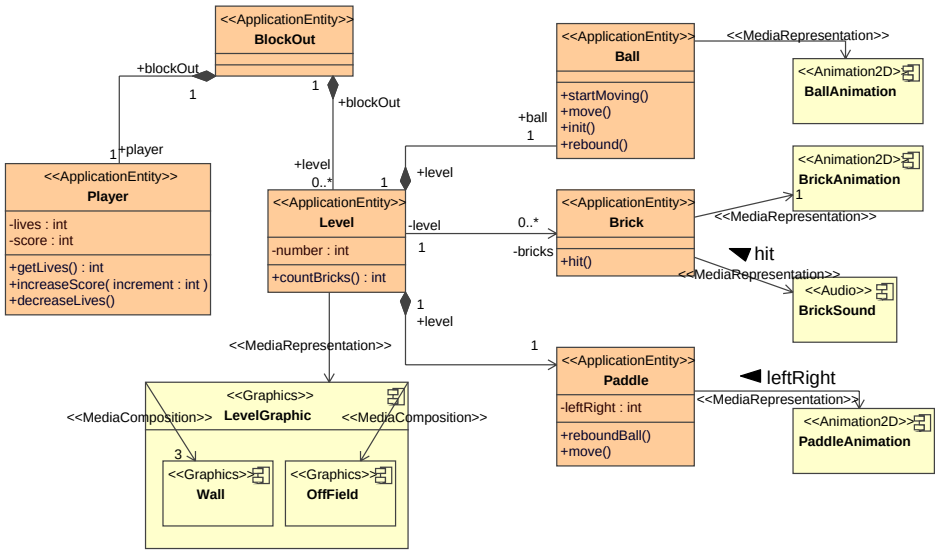
Application Structure: Application Entities Example



Application Structure: Application Entities Model Elements

- Analoguous 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 naviable (no arrows: bidirectional)
  - Aggregation or composition
- Generalizations between classes

Application Structure: Media Components Example



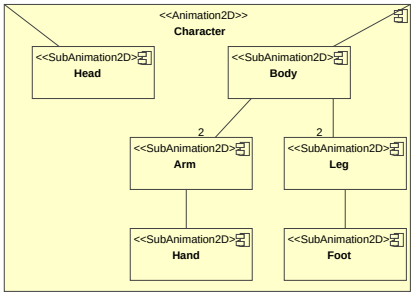
Application Structure: Media Components Model Elements (1)

- *Media component* is type of *2D-animation* (i.e. MovieClip in Flash), *3D-animation*, *audio*, *video*, *text*, *image*, *graphics*
- A 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
  - Specfied in the model by *Media-Representation* relationship between application entity and media component
  - Media-Representation may have multiplicities if an object is represented by multiple media components or if a media component represents multiple objects
  - Can be annotated with a specific attribute or operation represented by the media component, e.g. *PaddleAnimation* represents the value *leftRight* from *Paddle*

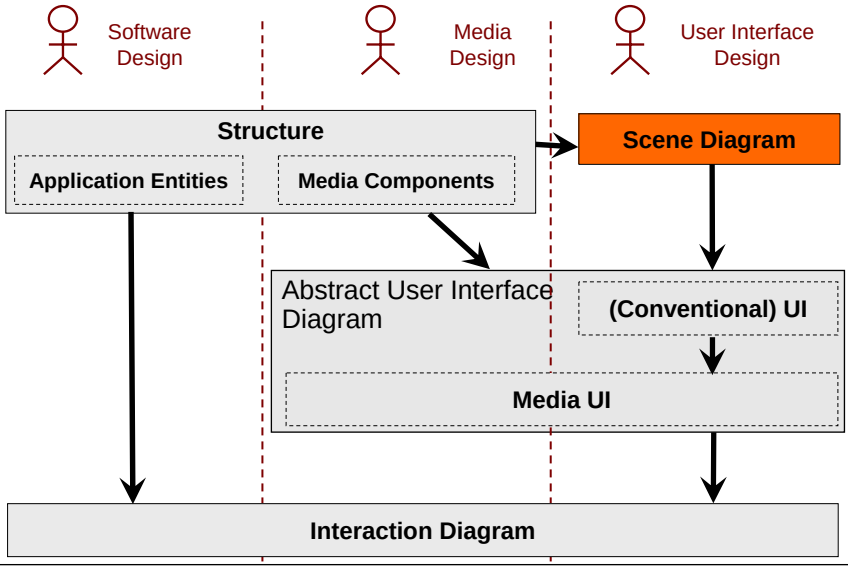
Application Structure: Media Components Model Elements (2)

- Inner structure of media components can be specified if necessary
  - **Only** necessary if application logic must access inner parts of the component
  - Additional example: the wheels of an animated racing car should turn when the car drives through a corner
- 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

- Hierarchy of Sub-Components
- Media-Composition my have multiplicity to specify e.g. a flexible number of children
- (In MagicDraw: Sterotypes *SubAnimation* and *MediaComposition* have to be added manually)



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)

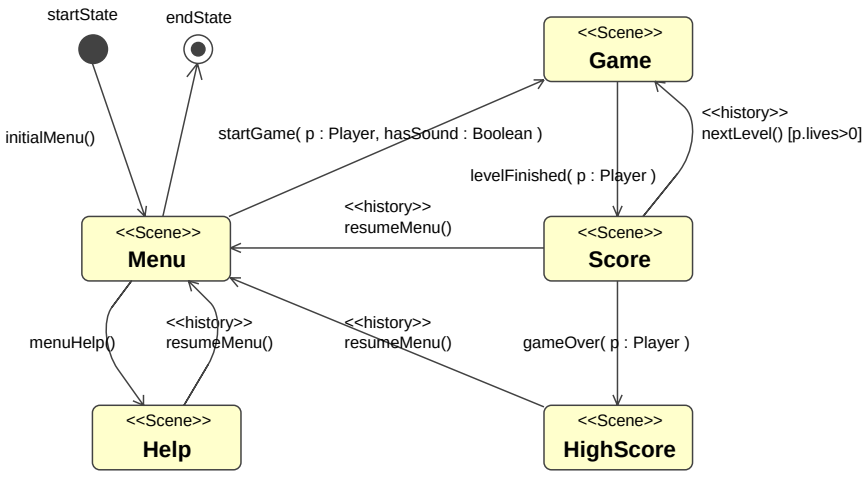
Parts:

- Scenes an Transitions between them

Notation:

- Adapted UML state charts

Scenes Example



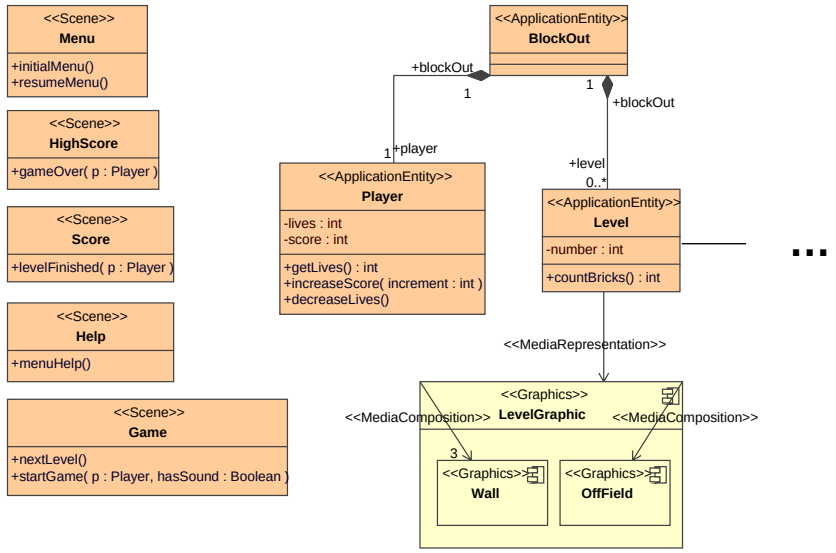
Scenes in the Scene Diagram

- *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*.

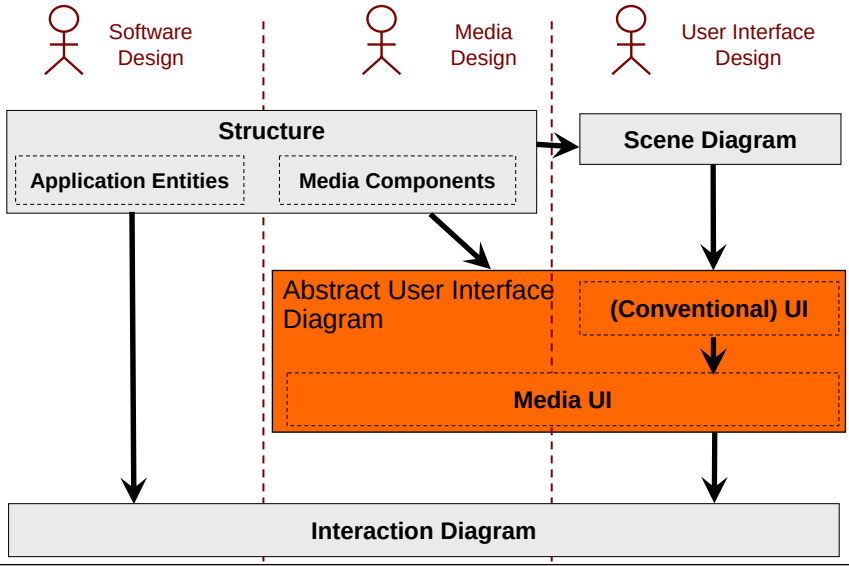
Scenes in the Application Structure Diagram

- In addition, scenes are modeled in the application structure diagram to specify their attributes and operations
  - Entry-Operations with parameters to initialize the scene
  - Exit-Operations which clean up the scene and invoke the subsequent scene
- Basically, scenes can be part of the structure diagram like application entities
  - Additional attributes and operations required for functionality of the scene
  - Scenes may be associated with media components
    - » Example: An application starts with a intro video. As the intro video does not represent an application entity, it is associated with a corresponding scene „Intro"
- Recommendation: If possible, try to separate scenes from application logic
  - Scenes can receive required objects through parameters of their eintry-operations
  - Like in the block-out example, there is a separate scene „Game" instead of defining „Level" as a scene

Application Structure Diagram enhanced with Scenes



Diagrams in MML



Abstract User Interface Diagram

Motivation:

- Platform independent specification of a scene‘s user interface in terms of abstract user interface components
- 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
- Temporal media components may cause additional events beside events from user interface components (e.g. termination of a video, collision of an animation)

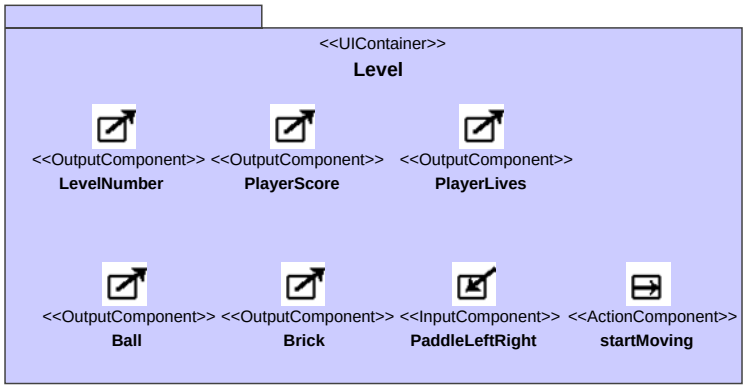
Parts:

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

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

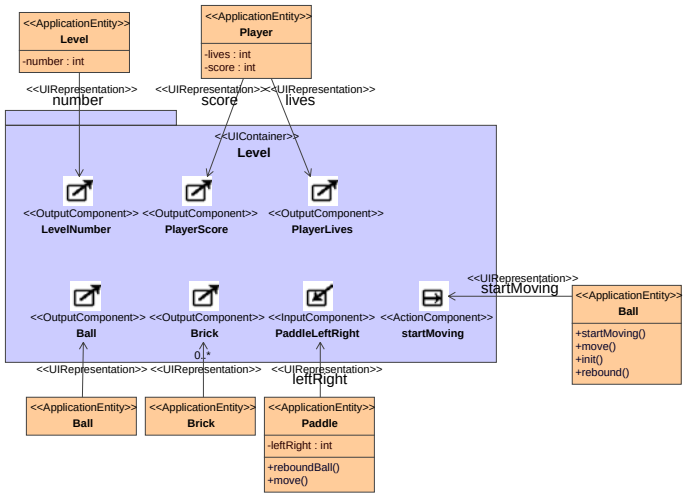
Abstract User Interface: User Interface Components Example for Scene *Game*



Abstract User Interface: User Interface Components Model Elements

- 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-Representations Example for Scene *Game*

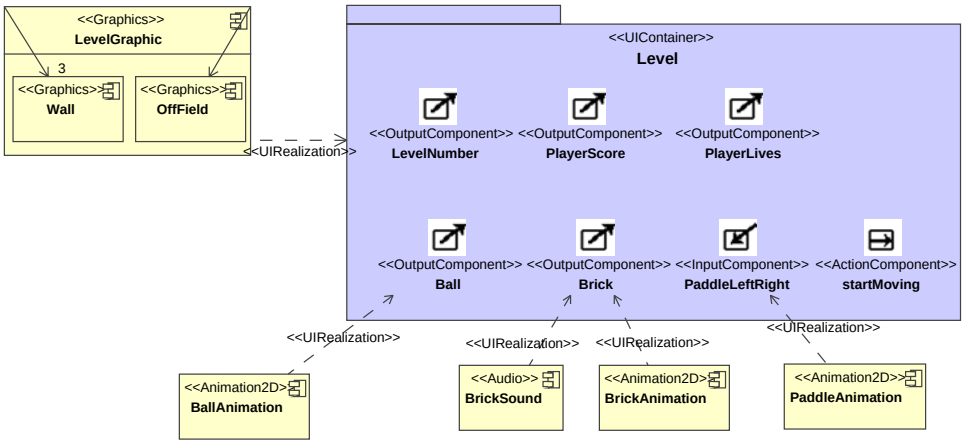


Abstract User Interface Model Elements

- UICs represent application logic (application entities) from the application structure diagram
- *UI-Representation* relationship analogous to Media-Representations in the application structure diagram
  - Can have multiplicities
  - Can specify an attribute or operation which is represented by the UIC

- Hint for usage of modeling tools (like MagicDraw):
  - Complete overall model is contained in repository („Containment-Tree“, „Explorer-Window“)
  - Each diagram is just a specific view on the overall model
  - One model element can be visible in multiple diagrams and multiple times within one digram
  - To reuse application entities within the user interface diagram, just drag them from the containment tree into the user interface diagram
  - Important conclusion: to delete a model element from the overall model, it must be deleted from the repository (not just delete it from a diagram!)

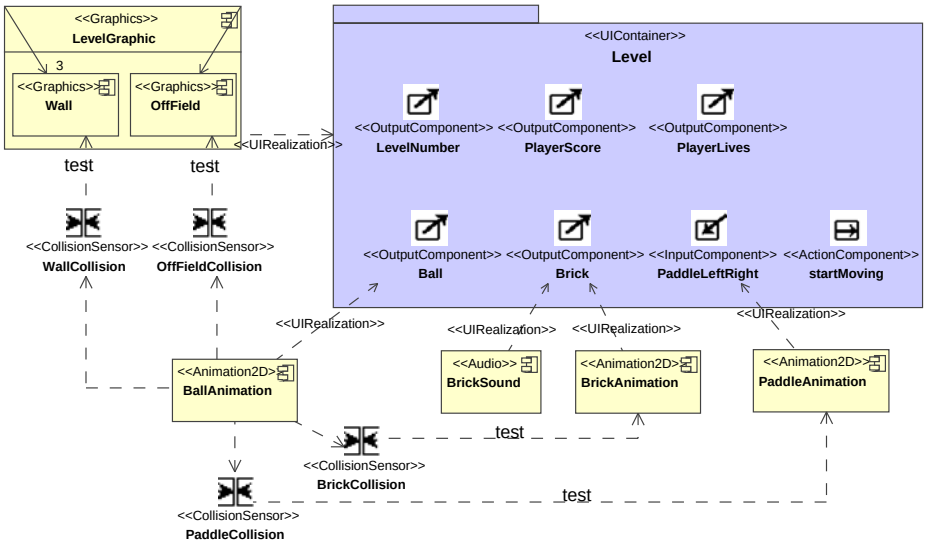
Abstract User Interface: UI-Realizations Example for Scene *Game*



Abstract User Interface: UI-Realizations
Model Elements

- 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.)

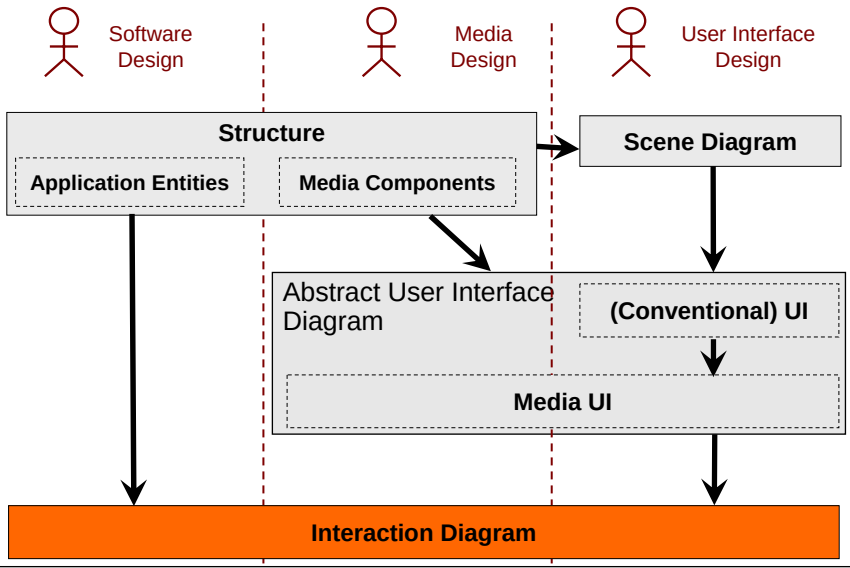
Abstract User Interface: Sensors
Example for Scene *Game*



Abstract User Interface: Sensors
Model Elements

- Temporal media components may cause additional events beside events from user interface components (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)

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

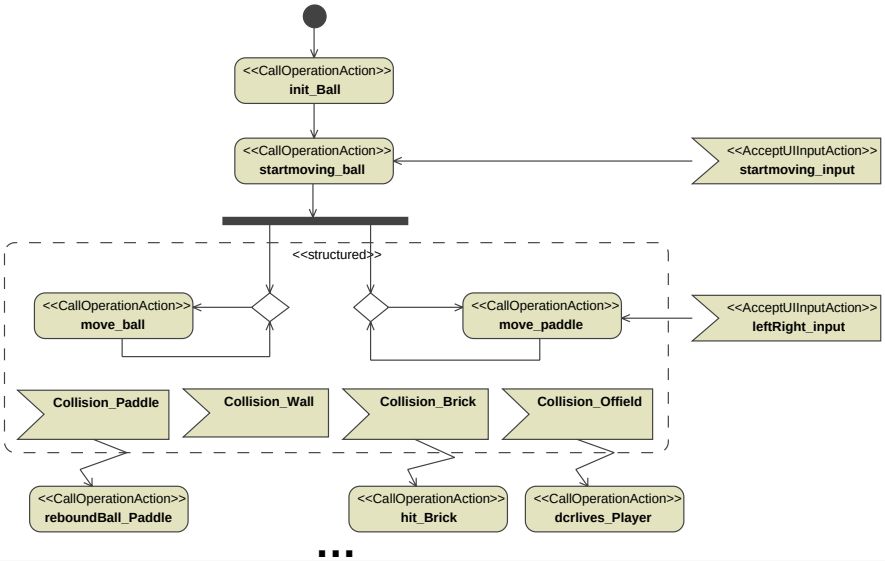
Parts:

- For each scene an activity

Notation:

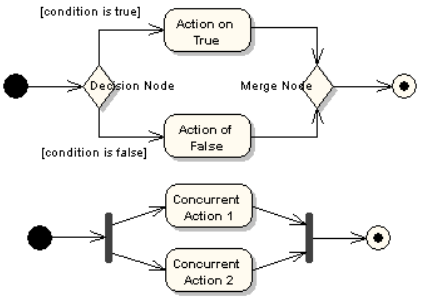
- UML activity diagram with limited set of actions and with references to AUIs and sensors from the media user interface

Interaction Diagram
Example for Scene *Game*

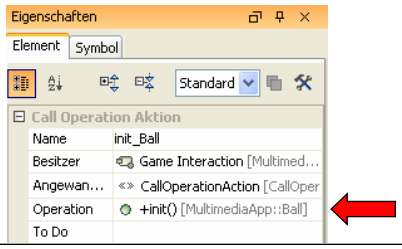


Interaction Diagram Model Elements (1)

- 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

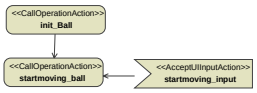


- Action in MML: Calls an operation from the structural model (all Actions in MML are from type *CallOperationAction*)
- In MagicDraw: Target operation is specified in the Action‘s property window (see figure: operation *init()* from application entity *Ball*)



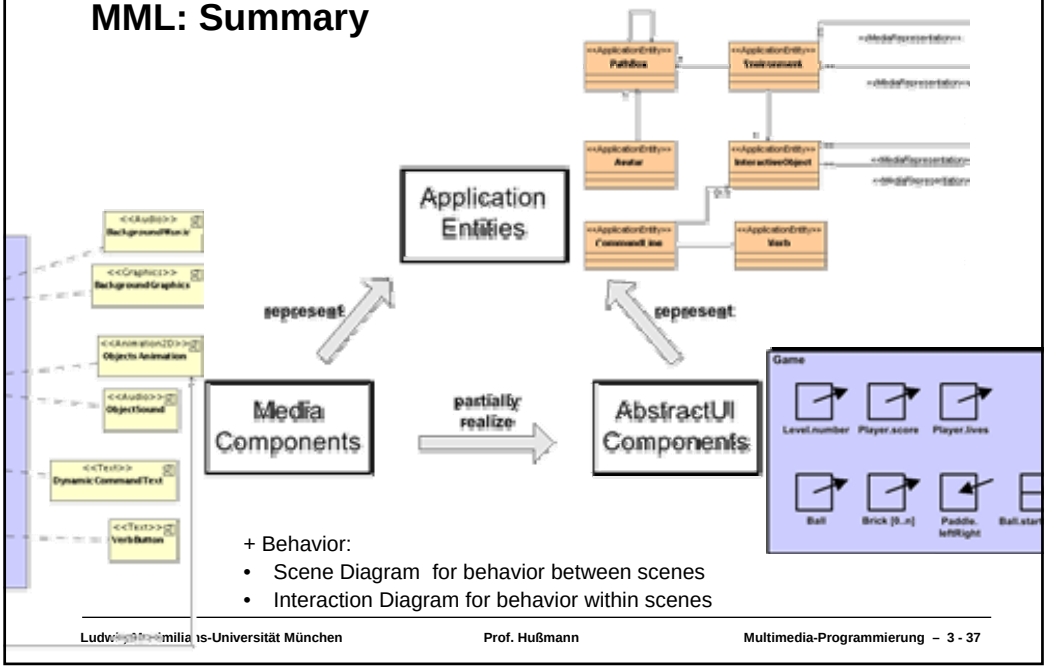
Interaction Diagram Model Elements (2)

- In addition, program flow is influenced by events from UICs or sensors
- Types of events (signals):
  - *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
- „OnEnterFrame()“ in Flash can be modeled using a time sensor
- By default, an action with multiple incoming flows waits until all preceding actions/events are executed



- An *interruption region* is used to specify that all its contained actions can be interrupted by an event (use “Strukturierter Aktivitätsknoten” in MagicDraw)

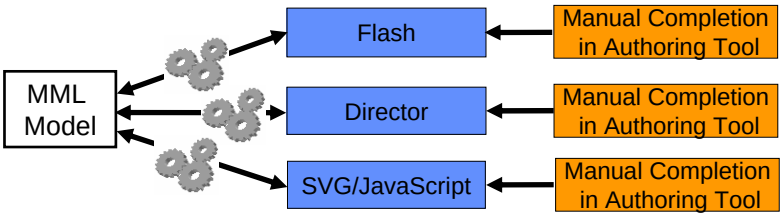
MML: Summary



+ 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

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’)