2 Development of multimedia applications

Animation as Attribute Modification

- Animation:
  - Modification of object attributes dependent on time / current frame
- Questions:
  - How to flexibly react on progress of time?
    - » Special events
  - How to program time-dependent code?
    - » Absolute computation of position
    - » Relative computation of position

Progress of Time as Event

- Most multimedia runtime systems have a notion of an event marking progress of time
  - Timer objects
  - Global clock
- ActionScript:
  - Special clip event **`EnterFrame`** is fired regularly at specified frame rate of the movie

Events in ActionScript

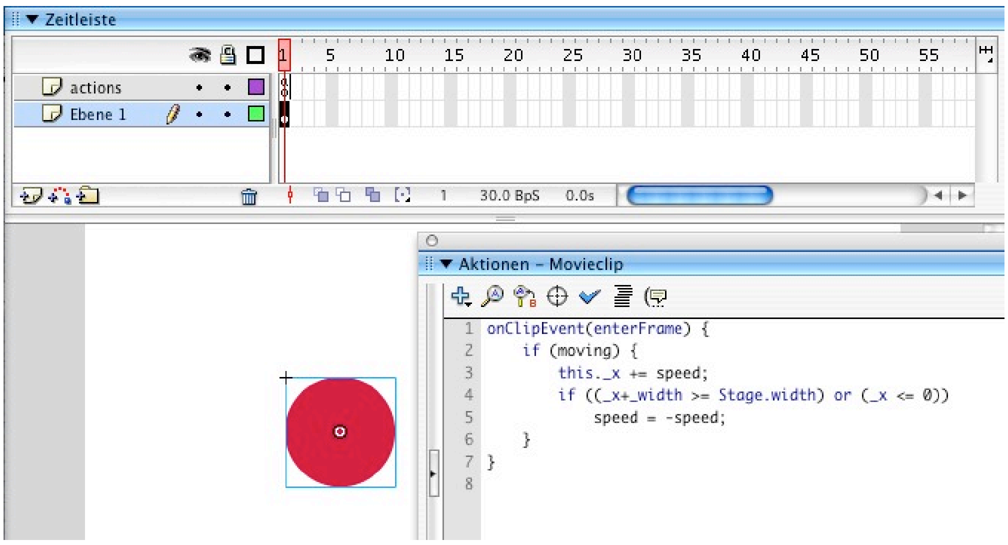
- Clip events (affecting a whole movie clip):
  - Load
  - Unload
  - EnterFrame
  - Mouse...
  - Key..
  - Data

`onClipEvent(...)`

- Interaction events (caused by specific interaction objects, e.g. buttons):
  - Press
  - Release
  - ReleaseOutside
  - RollOut, RollOver
  - DragOut, DragOver
  - KeyPress

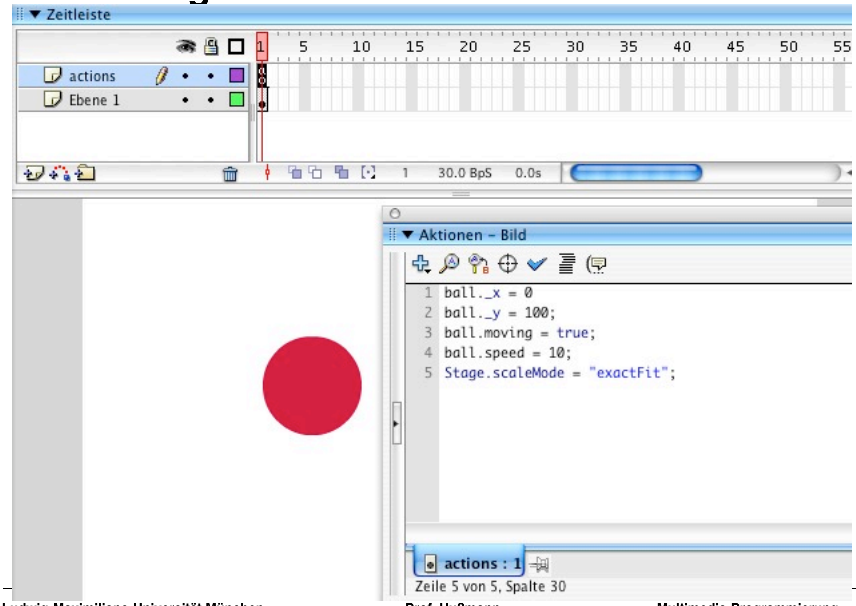
`on(...)`

Horizontal Movement with EnterFrame-Events



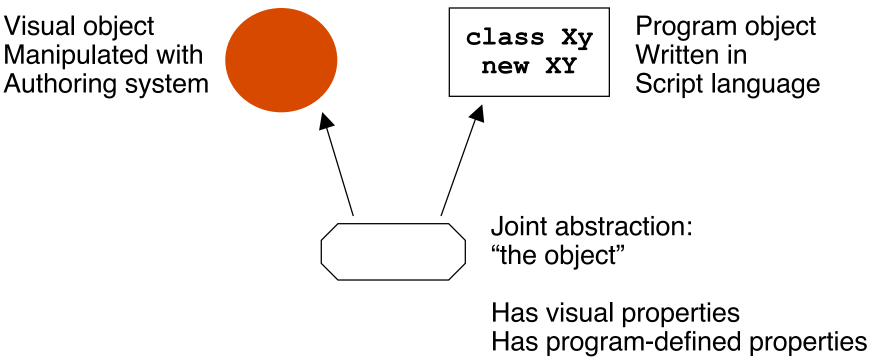
```
onClipEvent(enterFrame) {
    if (moving) {
        this._x += speed;
        if ((_x+_width >= Stage.width) or (_x <= 0))
            speed = -speed;
    }
}
```

“Main Program” for Horizontal Movement



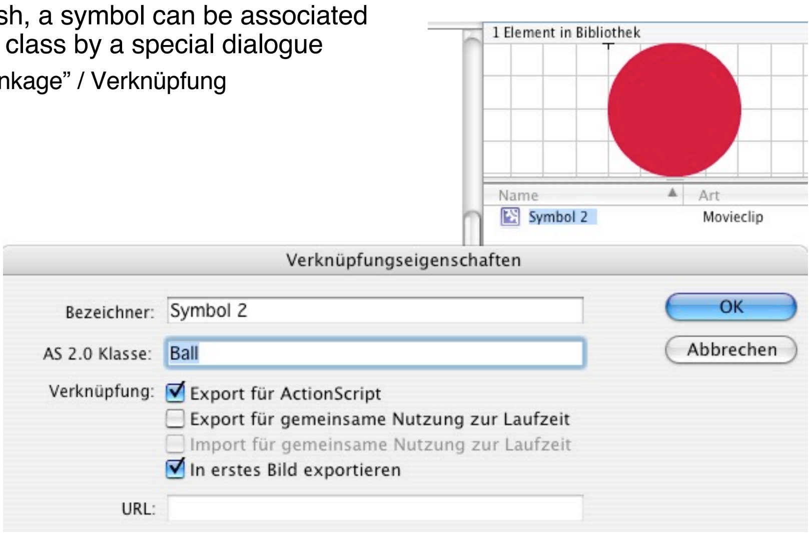
```
ball._x = 0
ball._y = 100;
ball.moving = true;
ball.speed = 10;
Stage.scaleMode = "exactFit";
```

Visual Objects and Program Objects



Flash: Linking AS2 Classes to Symbols

- In Flash, a symbol can be associated with a class by a special dialogue
  - “Linkage” / Verknüpfung



Bezeichner: Symbol 2

AS 2.0 Klasse: Ball

Verknüpfung:
- [✓] Export für ActionScript
- [ ] Export für gemeinsame Nutzung zur Laufzeit
- [ ] Import für gemeinsame Nutzung zur Laufzeit
- [✓] In erstes Bild exportieren

URL:

ActionScript 2 Class for Movement Example

```
class Ball extends MovieClip {
  public var speed:Number = 0;
  public var moving:Boolean = false;

  public function onEnterFrame() {
      if (moving) {
           this._x += speed;
           if ((_x+_width >= Stage.width) or (_x <= 0))
                speed = -speed;
      }
  }
}
```

Equivalent event handler declarations:
- attached to the object with generic keywords **`on`** and **`onClipEvent`**
- separate *callback* method (naming convention)

More powerful:
- listeners (see below)

Adding Vertical Movement

```
class Ball1 extends MovieClip {

  public var speed:Number = 0;
  public var jump:Number = 0;
  public var moving:Boolean = false;
  public var toRight = true;
  public var inLeftHalf:Boolean;

  public function onEnterFrame() {
      if (moving) {
           this._x += speed;
           if ((_x+_width >= Stage.width) or (_x <= 0)) {
                speed = -speed;
                toRight = !toRight;
           };
           inLeftHalf = (_x+_width)*2 <= Stage.width;
           if ((inLeftHalf && toRight) ||
           (!inLeftHalf && !toRight))
                _y -= jump;
           else
                _y += jump;
      }
  }
}
```

Absolute vs. Relative Movement Calculation

- Absolute calculation
  - Based on some base index
    - » Frame count, time, relative position on stage, ...
  - Base index to be provided by the programmer
    - » _currentframe, _totalframe etc. provide statically defined information
  - “Save” in terms of predictibility of the effect

- Relative calculation
  - Based on most recent frame (“differential programming”)
  - Often easier (see example)
  - More flexible for changing situations
  - Problem: Rounding errors and other algorithmic problems may lead to unexpected effects (see example)

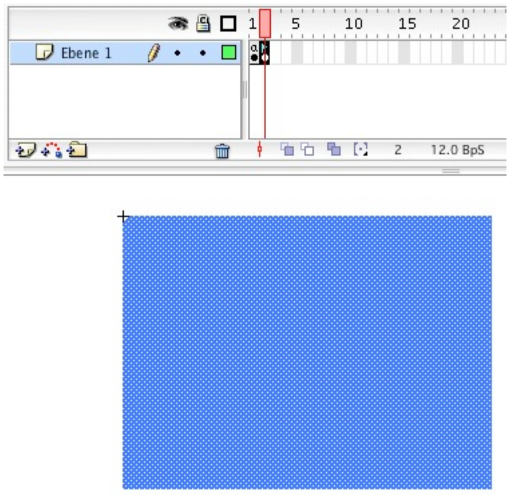
2 Development of multimedia applications

What’s Specific for an Animated (Flash) Interface?

- Traditional user interface elements:
  - Buttons, Textfields, Menus, ...
  - All available also in Flash and other modern multimedia interface tools
- Animation in user interfaces:
  - Graphical feedback illustrating program actions
    - » E.g. direction of money transfer, strong warning: animation clips
  - Direct feedback “on touching”
    - » E.g. change of graphical representation on “mouse over”
- Direct interaction:
  - Drag and drop
  - Drawing-like actions
- Everything (in principle) realisable also by “normal” programming languages! (But often much more complex.)

Example: Highlighting a Region on “RollOver”

- Graphical element with AS event handler for “RollOver” event
  - E.g. changing the colour of a box
- “Traditional” solution with the Flash authoring tool:
  - Create a symbol with different key frames
  - Create an instance with an event handler switching between key frames



Event Handler for Frame Switching

```
on(rollOver) {
  gotoAndStop("on");
}
on(rollOut) {
  gotoAndStop("off");
}
```

"**`on`**" and "**`off`**" are labels for the key frames of the symbol.

Not to be forgotten: **`stop()`** in first frame.

Flash Pattern: Graphical Response

- **Problem**: Dependent on some application-internal condition, we would like to show the user what the current status is, by selection among different graphical representations.
- **Solution:**
  - Create a MovieClip object and create different key frames showing the different graphical representations of status information. If the information is not to be shown sometimes, one key frame may remain empty.
  - Add a **`stop();`** action to the first key frame.
  - Optionally, assign labels to the key frames.
  - Place the MovieClip object on the stage
  - Show various status information by "gotoAndStop()" to the MovieClip object.
- **Examples**:
  - Realisation of the generic pre-defined Button class
  - Quiz example from ActionScript 2.0 Dictionary, pp. 8 ff.

A More Object-Oriented Solution

- Problems with the “traditional” solution:
  - Four different regions (with different highlighting colours) require four symbols
  - Event handling code has to be attached to *instance* of MovieClip symbol
  - Event handling code is duplicated
- The Macromedia partial solution:
  - Introduction of the special “Button” class
- A Programmer’s solution (next few slides):
  - Create a reusable class for a highlightable region
  - Make the color into a parameter settable from outside

Reusable Highlighting Color Block

```
class ColorBlock extends MovieClip {

  private var myColor:Color;
  public var myOnRgb:Number;

  public function onLoad() {
     myColor = new Color(this);
  }

  public function onRollOver() {
     gotoAndStop("on");
     myColor.setRGB(myOnRgb);
  }

  public function onRollOut() {
     gotoAndStop("off");
     myColor.setRGB(0xffffff);
  }
}
```

Used built-in technology:

`Color` object controls the color of the movie clip.

Constructor assigns the new object to the given movie clip.

`setRGB` function actually changes the color.

Creating Instances of the Reusable Symbol

- There is *one* symbol with several instances (example: lo_mc, ro_mc, lu_mc, ru_mc)
- The symbol defines the graphical shape with irrelevant color.

- Initialisation code:

```
lo_mc.myOnRgb = 0xff0000; //red
ro_mc.myOnRgb = 0x0000ff; //blue
lu_mc.myOnRgb = 0x00ff00; //green
ru_mc.myOnRgb = 0xffff00; //yellow
```

Creating a “Graphically Enhanced” User Interface

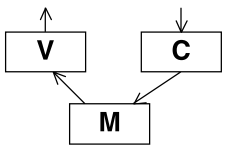
- Traditional programming
  - Example: Account with credit and debit function

- Additional “multimedia” features:
  - Auto-highlighting buttons
  - Visualization of money transfer direction
  - Visualization of “low” warning

The Account Class

```
class Account {

  var saldo:Number = 0;
  var num:Number;

  function Account(accnum:Number) {
      num = accnum;
  }

  function debit(n:Number) {
      saldo -=n;
  }

  function credit(n:Number) {
      saldo +=n;
  }

  function getNumber():Number {
      return (num);
  }

  function getSaldo():Number {
      return (saldo);
  }
}
```

Model-View-Controller (MVC) Paradigm



- Model:
  - Business model, mostly independent of user interface
  - Observable by arbitrary objects (application of *Observer* pattern)
- View:
  - Representation on user interface
  - Observes the model
  - Asks required data from the model
- Controller:
  - Modifies values in the model
  - Is driven by user interactions, therefore bound to elements of interface
  - Handles events mainly by calling methods of the model

Predefined Event Dispatcher

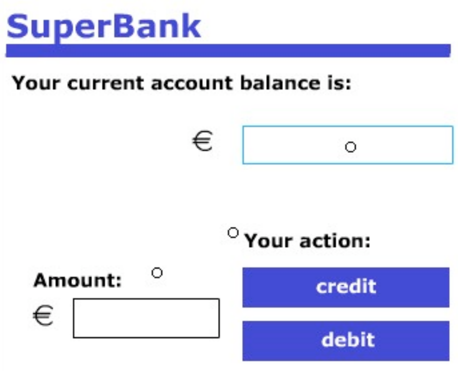
- Code base for library of predefined ActionScript classes:
  - In “Configuration/Classes” subdirectory
  - Contains readable ActionScript code (often undocumented)
- “mx” subdirectory:
  - Library functions for advanced use of ActionScript
  - E.g. “mx.events. ...”
  - Example class: **`EventDispatcher`**
- Usage by “import” statement as in Java
  - E.g. **`import mx.events.EventDispatcher;`**

Model: Account Class with Event Dispatching

```
import mx.events.EventDispatcher;

class Account extends EventDispatcher {

  var saldo:Number = 0;
  var accNum:Number;

  function Account(an:Number) {
      accNum = an;
  }

  function debit(n:Number) {
      if (n < 0) return;
      saldo -=n;
      if (n <> 0)
            dispatchEvent({type:"saldoLower"});
  }

  function credit(n:Number) {
      if (n < 0) return;
      saldo +=n;
      if (n <> 0)
            dispatchEvent({type:"saldoHigher"});
  } ...
}
```

View: User Interface Design

- Main output form is a (dynamic) text field
- However:
  - Text fields cannot carry ActionScript code
  - Text field cannot be easily associated with AS class
- How can we stay object-oriented?
- Idea: Add a new function to the text field object...



Extending a TextField Object

- `saldo_txt` is a TextField object generated in the authoring tool
- Extension code (in main timeline):

```
saldo_txt.update = function(){
  var saldo: Number = myAccount.getSaldo();
  saldo_txt.text = saldo;
  if (saldo < 0)
     lowWarning_mc.gotoAndPlay("startAnim");
  else
     lowWarning_mc.gotoAndStop("stopAnim");
}
```

Connecting View to Model

- Using EventDispatcher
- Event handling code for updating view

```
var myAccount:Account = new Account(1234);
myAccount.addEventListener
  ("saldoLower",saldoLowerHandler);
myAccount.addEventListener
  ("saldoHigher", saldoHigherHandler);

function saldoLowerHandler(eventObj){
  debit_mc.gotoAndPlay("startAnim");
  saldo_txt.update();
}

function saldoHigherHandler(eventObj){
  credit_mc.gotoAndPlay("startAnim");
  saldo_txt.update();
}
```

Controller: User Event Handling

- Using Flash’s built-in `Button` class makes highlighting easy.
- Event handling code (example “credit”, “debit” is similar):

```
on (release) {
  var amount:Number = Number(amount_txt.text);
  if (isNaN(amount) or (amount < 0)) {
     amount_txt.text += "?";
  }
  else {
     myAccount.credit(amount);
  }
}
```