7 Software Engineering Techniques for Multimedia Software

7.1 Design Patterns: The Idea
7.2 Patterns for Multimedia Software (contd.)
7.3 Gang-of-Four Patterns Applied to Multimedia
7.4 Modeling of Multimedia Applications
Classification Space

Time usage:
- Still picture
- Linear progress
- Interaction dependent progress

Space usage:
- Static layout
- Scenes
- Scenes & objects
- Fully dynamic

Interactivity:
- Fully automatic (passive)
- Confirmations & questions (reactive)
- Interactive controls (proactive)
- Creation & dragging of objects (directive)
Multimedia Development Pattern: Time Container Algebra

- Presentation is built from atomic parts (processes) each of which is executed in a *time container*.
- Time containers are composed by algebraic operations: sequential composition, parallel composition, repetition, mutual exclusion, synchronisation options
- Time usage: Linear progress
- Space usage: Scenes or scenes&objects
- Low interactivity
- Examples:
  - SMIL body: seq, par, excl
  - Animations class of “JGoodies” animation framework for Java
  - Sequence of frames and parallelism of layers in Flash
Various Representations of a Single Concept

Component \( r_1 = \ldots; \)
Animation \( \text{frame1} = \ldots; \)
Animation \( \text{frame2} = \ldots; \)
Animation \( \text{all} = \)
\( \text{Animations.\text{sequential}}(\)\[\text{new Animation[]}\{\text{frame1, frame2}\});\)

---

XML

Java

Authoring Tool (Flash-like)
Flash Pattern: Start Frame Code

• **Problem**: A Flash movie needs to carry out some ActionScript code which cannot be easily defined in a local, object-oriented style
  - Creation of objects on an application-global scale
  - Invocation of methods defined in external “.as” files
  - Assignment of methods to visible objects instantiated from the standard library (e.g. TextField)

• **Solution**:
  - Keep the “global code” in the main timeline.
  - Add a separate layer (e.g. “code” or “actions”) to the main timeline.
  - Add all “global” code to frame 1 of the newly created layer of the main timeline.
  - Advantage: There is just one place where all global code can be found.

• **Examples**:
  - Plenty found in literature
7 Software Engineering Techniques for Multimedia Software

7.1 Design Patterns: The Idea
7.2 Patterns for Multimedia Software (contd.)
7.3 Gang-of-Four Patterns Applied to Multimedia
   Factory Method
   State
7.4 Modeling of Multimedia Applications

Literature:
W. Sanders, C. Cumaranatunge: ActionScript 3.0 Design Patterns, O’Reilly 2007
Creation Pattern Example: Factory Method

• Situation:
  – Families of products which behave similarly
    » Same interface
  – Example: Different kinds of players, weapons etc. in a game

• Motivation:
  – Keep code easy to change
    » Typical change: Adding a new member of the family
  – Decouple *using* the products from *creating* the products
  – Code creating a product shall not know about the range of possible products
    » Shall not have access to the product subclasses

• Idea:
  – Provide method with the only purpose of creating products (factory method)
GoF Creation Pattern: Factory Method

- **Name:** Factory Method  
  (dt.: Fabrikmethode, auch: Virtueller Konstruktor)
- **Problem:**  
  - Choose at creation time between variants of a product
- **Solution:**

```
Product
{abstract}

Concrete Product1

Concrete Product2

Creator
create (...): Product

Client

<<create>>

<<create>>

<<use>>
```
Example for Factory Method (1)

• Variants of products:
  – Ships:
    » Hero ship
    » Alien ship
  – Weapons:
    » Hero weapon
      • Cannon
    » Alien weapon
      • Cannon
      • Mine

• We want to keep the code extensible for new ship and weapon types

“Open-closed principle”:
Open for extensions, closed for code modification

Example: Sanders/Cumaranatunge
Example for Factory Method (2)

```
Ship
   /\ \
  /  \  \\
HeroShip   AlienShip

<<create>>

Weapon
   /\ \\
  /  \\
HeroWeapon   AlienWeapon

<<create>>

ShipCreator

<<create>>

Projectile
   /\ \\
  /  \\
HeroCannonBall   AlienCannonBall   AlienMine

<<create>>
```
Example for Factory Method (3)

```javascript
package ships {

    import flash.display.Sprite;
    import flash.events.*;

    // ABSTRACT Class (should not be instantiated)
    internal class Ship extends Sprite {

        internal function setLoc(xLoc:int, yLoc:int):void {
            this.x = xLoc;
            this.y = yLoc;
        }

        // ABSTRACT Method (must be overridden in a subclass)
        internal function drawShip():void {
        }

        // ABSTRACT Method (must be overridden in a subclass)
        internal function initShip():void {
        }
    }
}
```
Example for Factory Method (4a)

package ships {

    import flash.display.*;
    import weapons.HeroWeapon;
    import flash.events.***;

    internal class HeroShip extends Ship {

        private var weapon:HeroWeapon;

        override internal function drawShip():void {
            graphics.beginFill(0x00FF00); // green color
            graphics.drawRect(-5, -15, 10, 10);
            graphics.drawRect(-12, -5, 24, 10);
            graphics.drawRect(-20, 5, 40, 10);
            graphics.endFill();
        }

        ...
    }
}
Example for Factory Method (4b)

... 

override internal function initShip():void {
    weapon = new HeroWeapon();
    this.stage.addEventListener(MouseEvent.MOUSE_MOVE,
        this.doMoveShip);
    this.stage.addEventListener(MouseEvent.MOUSE_DOWN,
        this.doFire);
}

protected function doMoveShip(event:MouseEvent):void {
    this.x = event.stageX;
    event.updateAfterEvent(); // process this event first
}

protected function doFire(event:MouseEvent):void {
    weapon.fire(HeroWeapon.CANNON,
        this.stage, this.x, this.y - 25);
    event.updateAfterEvent(); // process this event first
}
Example for Factory Method (5)

```java
package {

    import flash.display.*;
    import flash.text.*;
    import ships.*;

    public class Main extends MovieClip {

        public function Main() {
            // show instructions

            var shipFactory:ShipCreator = new ShipCreator();
            shipFactory.addShip(ShipCreator.HERO, stage,
                               stage.stageWidth/2, stage.stageHeight-20);
            for (var i:Number = 0; i < 5; i++) {
                shipFactory.addShip(ShipCreator.ALIEN, stage, 120 + 80 * i, 100);
            }
        }
    }
}
```
Example for Factory Method (6a)

package ships {

    import flash.display.Stage;

    public class ShipCreator {

        public static const HERO : uint = 0;
        public static const ALIEN : uint = 1;

        public function addShip(cShipType:uint, target:Stage, xLoc:int, yLoc:int):void {

            var ship:Ship = this.createShip(cShipType);
            ship.drawShip();
            ship.setLoc(xLoc, yLoc);
            target.addChild(ship);
            ship.initShip();
        }

        ...
    }
}
Example for Factory Method (6b)

...  

// concrete factory method
private function createShip(cShipType:uint):Ship {
    if (cShipType == HERO) {
        trace("Creating new hero ship");
        return new HeroShip();
    } else if (cShipType == ALIEN) {
        trace("Creating new alien ship");
        return new AlienShip();
    } else {
        throw new Error("Invalid kind of ship specified");
        return null;
    }
}
Test for Encapsulation

```csharp
public function Main() {
    ...
    var testShip = new HeroShip();
    ...
}
```

Compiler-Fehler:
1180: Aufruf einer möglicherweise undefinierten Methode HeroShip.
Test for Extensibility (1)

• How to add a new weapon?
• HeroShip.as:
  override internal function initShip():void {
    weapon = new HeroWeapon();
    this.stage.addEventListener
    (MouseEvent.MOUSE_MOVE, this.doMoveShip);
    this.stage.addEventListener(MouseEvent.MOUSE_DOWN, this.doFire);
    var newweapon = new NewWeapon();
    newweapon.fire(NewWeapon.NEW, this.stage, this.x, this.y - 50);
  }

• New classes added *(without modification of existing code!)*
  – NewWeapon.as
    » The new kind of weapon
    » Concrete creator for bullets, derived from abstract creator Weapon
  – NewBullet.as
    » The bullet fired by the new kind of weapon
    » Concrete product, derived from abstract product Projectile
package weapons {

    public class NewWeapon extends Weapon {

        public static const NEW :uint = 3;

        override protected function createProjectile(cWeapon:uint):Projectile {
            if (cWeapon == NEW) {
                trace("Creating new bullet");
                return new NewBullet();
            } else {
                throw new Error("Invalid kind of projectile");
                return null;
            }
        }
    }
}
Test for Extensibility (3)

```java
package weapons {

    internal class NewBullet extends Projectile {

        override internal function drawProjectile():void {
            graphics.beginFill(0xFF0000);
            graphics.drawCircle(0, 0, 15);
            graphics.endFill();
        }

        override internal function arm():void {
            nSpeed = -15; // set the speed
        }
    }
}
```

• Methods `drawProjectile()` and `arm()` are called in method `fire()` of abstract class `Weapon`
  – Idea of *Template Method* pattern
7 Software Engineering Techniques for Multimedia Software

7.1 Design Patterns: The Idea
7.2 Patterns for Multimedia Software
   (contd.)
7.3 Gang-of-Four Patterns Applied to Multimedia
   Factory Method
   State
7.4 Modeling of Multimedia Applications

Literature:
W. Sanders, C. Cumaranatunge: ActionScript 3.0 Design Patterns,
O’Reilly 2007
GoF Structural Pattern: State

• **Name:** State
• **Problem:**
  
  Flexible and extensible technique to change the behavior of an object when its state changes.
• **Solution:**

```plaintext
Context
  request()
  setState (s: State)

State
  {abstract}
  handle() {abstract}

State1
  handle() 

State2
  handle()
... 
```

calls state.handle
Example for State (1)

```
State

startPlay()
stopPlay()

VideoWorks
```

```
StopState
startPlay()
stopPlay()

PlayState
startPlay()
stopPlay()
```
Example for State (2)

interface State {
    function startPlay(ns: NetStream, flv: String): void;
    function stopPlay(ns: NetStream): void;
}

class PlayState extends State {
    public function startPlay(...): void {
        trace("Already playing");
    }
    public function stopPlay(...): void {
        ns.close();
        videoWorks.setState(
            videoWorks.getStopState());
    }
}

class StopState extends State {
    public function startPlay(...): void {
        ns.play(flv);
        videoWorks.setState(
            videoWorks.getPlayState());
    }
    public function stopPlay(...): void {
        trace("Already stopped");
    }
}
Test for Extensibility

• Adding a “pause” state
• First step: Change the State interface
  function doPause(ns:NetStream):void;
  – Compiler checks completeness of transitions
• (1044: Schnittstellenmethode doPause in Namespace State nicht durch
  Klasse PlayState implementiert.)
7 Software Engineering Techniques for Multimedia Software

7.1 Design Patterns: The Idea
7.2 Patterns for Multimedia Software
    (contd.)
7.3 Gang-of-Four Patterns Applied to Multimedia
7.4 Modeling of Multimedia Applications

Literature:
Model-Driven Development

• Development process with models as core assets
  – "Model" = 'a simplified image of a system'

• 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
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

- Structure Diagram
  - Application Entities
  - Media Components

- Scene Diagram

- Abstract User Interface Diagram

- (Conventional) UI

- Media UI

- Interaction Diagram
Example: Application structure for Break Out Game Application

Inner Structure

1. BlockOut
   - lives : int
   - score : int
   + getLives() : int
   + decreaseLives() (4)
   + increaseScore() (4)

2. Player
   - lives : int
   - score : int
   + getLives() : int
   + decreaseLives() (4)
   + increaseScore() (4)

3. Level
   - number : int
   + countBricks() : int

4. Ball
   - startMoving()
   - move()
   - init()
   + rebound()

5. Brick
   - hit()

6. Paddle
   - leftRight
   + reboundBall()

7. «Animation> PaddleAnimation
8. «Animation> BrickAnimation
9. «Sound> BrickSound
10. «Animation> BallAnimation

Media Representation

Media Component
Example: Scenes for Break Out Game Application

- **Scene**: initialMenu
  - EntryOperation: startGame(p:Player, hasSound:Boolean)

- **Scene**: Menu
  - EntryOperation: levelFinished(p:Player)
    - Transition: [p.lives > 0]
      - Transition: <<history>> nextLevel()

- **Scene**: Help
  - Transition: menuHelp
  - Transition: <<history>> resumeMenu

- **Scene**: Score
  - Transition: gameOver(p:Player)
  - Transition: <<history>> resumeMenu

- **Scene**: Game
  - EntryOperation: Game

- **Scene**: Highscore
  - Transition: <<history>> resumeMenu
Example: Abstract User Interface for Scene ‘Game’

- **UI Container**
- **Edit Component**
- **Action Component**
- **Output Component**
- **Assigned Class/Property/**
- **Multiplicity**

**Game**

- **LevelNo**
  - `{Level.number}`
- **Score**
  - `{Player.score}`
- **Lives**
  - `{Player.lives}`

- **Ball**
  - `{Ball}`
- **Bricks [0..n]**
  - `{Brick}`
- **Paddle**
  - `{Paddle.leftRight}`
- **Start**
  - `{Ball.startMoving}`

---

Ludwig-Maximilians-Universität München
Prof. Hußmann
Multimedia-Programmierung – 7a – 33
Example: Media User Interface for Scene ‘Game’

UI Realization

Sensor

AUIs without specific realization
Example: Interaction diagram for Scene ‘Game’

- **init (ball)**
- **startMoving (ball::)**
- **move (ball::)**
- **move (paddle::)**
- **Paddle**
- **Wall**
- **Brick**
- **OffField**

CallOperation Action

Object (property of the scene)

UIInputEvent

Ball.start Moving

Paddle. leftRight

Sensor Event

...
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?

![Diagram showing integration of authoring tools into model]

**MML Model**

- *Flash*
- *Director*
- *SVG/JavaScript*

**Manual Completion in Authoring Tool**

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*
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?