Content

1. Overview
2. The running example: Ludo
3. Use case description
4. Object oriented analysis with story boards
5. Test derivation
6. Derivation of design and implementation
7. Validation
Fujaba Graph Rewriting Tool

- http://www.fujaba.de/
- http://www.fujaba.de/no_cache/publications.html
Overview

Story Driven Modeling with Graph Rewriting:

Steps:
- Textual use case description
- Story Boarding (OOA) (Test specification)
- Class diagram derivation (OOD)
- Behavior derivation (Coding)
- Code generation
- Validation (Testing)

Features:
- Use Case Driven
- Model Driven
- Iterative
- Test Driven Development
42.2. The running example: Ludo

- Development of an interpreter for a language (here Ludo game)
42.3. Use case diagrams (Rpt.)

Requirements elicitation as usual:
- Use case diagrams for overview
Textual scenario descriptions:

- focus on scenarios
- several scenarios per use case
- focus on one example situation at a time
- use concrete names
SDM approach is based on noun-verb-analysis:

- analyse the text scenarios
- nouns become *objects*
- verbs become *method invocations* or *links*
- ...

Start Situation: __________
________________________
________________________

p:Player       d:Dice
l:Ludo         s:Step
42.4 Object oriented analysis with story boards

- use case execution is modeled by one method invocation
- drawn as collaboration message
- multiple scenarios for one use case call the same method (but in different situations)
- this method implements the use case
- use case ↔ method mapping enables tracebility
- step descriptions may become implementation comments

Actor step 1: _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ 1: _ _ _ _ _ _ _ _
Relations in a use case are mapped to method calls

- **uc1 <<uses>> uc2** → method uc1() may call method uc2()
- **uc1 <<includes>> uc2** → uc1() always calls uc2()
- **uc2 <<extends>> uc1** → uc1() provides extension points / call backs. uc2() may subscribe for such a call back
Object oriented analysis with story boards in Fujaba

- Outlining method behavior in concrete example situations:

  - <<create>> and <<destroy>> markers
  - := attribute assignments
  - recurring objects without class name
    first time on stage with class name
    (change of perspective)
  - collaboration messages
  - alternatively sequence diagrams

```
Step 1: ________________________________
        ________________________________
```

```
<<create>> and <<destroy>>
markers

:= attribute assignments

recurring objects without class
name
first time on stage with class name
(change of perspective)

collaboration messages

alternatively sequence diagrams
```
Object oriented analysis with story boards

Result situation:

- models resulting object structure
- used for testing
Derivation of Class Diagrams

Collect the types from the story boards:

- Classes
- Associations
- Attribute declarations
- Method declarations
Derivation of Class Diagrams (cont.)

- Class diagram derivation is straightforward
- Semi-automatic tool support by Fujaba
- Intermediate story board step results in much better domain level class diagrams
- Code generation for class diagrams
- Story boards are appropriate for the analysis and discussion of behavior
- Story boards also useful during refinement and coding
- Story boards may serve as test specifications
- Story boards may drive the implementation
42.5. Test Derivation

- Scenarios → JUnit Tests
- start situation → setup code
- invocation → invocation
- result situation → code that checks object structure equivalence
Scenarios → JUnit Tests, start situation → setup code and fixture

Start Situation: Tom rolled a 3 and selects counter 1 for moving

```
class TestMoveUsual implements TestCase {
    private Player tom;
    private Die theDie;
    private Counter counter1;
    ...
    void setUp () {
        tom = new Player ();
        theDie = new Die ();
        theDie.setV (3)
        tom.setDie (theDie);
        counter1 = new Counter ();
        tom.addToCounters (counter1);
        ...
    }
```

Test Derivation (cont.2)

- Scenarios → JUnit Tests, start situation → setup code

Invocation: counter 1 is moved

```java
class TestMoveUsual implements TestCase {

    ... 

    void testMoveUsual () {
        this.counter1.move();
    
    
    ... 
```

counter1 1: move ()
Test Derivation (cont.3)

- Scenarios → JUnit Tests, start situation → setup code

Result Situation: the die is counted down to zero and counter 1 reached field 4

```java
class TestMoveUsual implements TestCase {
    void testMoveUsual () {
        this.counter1.move();
        assertTrue (tom.getDie() == theDie);
        assertTrue (theDie.getV() == 0);
        assertTrue (counter1.getPlayer () == tom);
        assertTrue (counter.getAt () == f4);
    }
}
```

Result Situation: the die is counted down to zero and counter 1 reached field 4
Test Derivation (cont.4)

- more complex result situations work, too (see later)
- start situation, invocation, result situation $\rightarrow$ JUnit tests
- steps may be exploited, too, cf. [SCESM05]
- analysis scenarios $\leftrightarrow$ tests
- test driven software development
42.6 Derivation of the Implementation

- combine story boards to rule diagrams [SCESM04]
- assign execution semantics
- code generation
### Story Pattern Elements:

#### Variables:

<table>
<thead>
<tr>
<th>v : Class</th>
<th>«create»</th>
<th>v : Class</th>
<th>«destroy»</th>
<th>v : Class</th>
<th>«create»</th>
<th>v : Class</th>
<th>«create»</th>
<th>v : Class</th>
<th>«destroy»</th>
</tr>
</thead>
<tbody>
<tr>
<td>v</td>
<td></td>
<td>v</td>
<td></td>
<td>v</td>
<td></td>
<td>v</td>
<td></td>
<td>v</td>
<td></td>
</tr>
</tbody>
</table>

#### Attributes:

| attr \(<cmpOp>\) value
| attr := value |

#### Constrains:

{ \(<boolExpr>\) }  
{first}  
{last}

#### Links:

<table>
<thead>
<tr>
<th>assoc</th>
<th>assoc[&lt;key&gt;]</th>
</tr>
</thead>
<tbody>
<tr>
<td>«create»</td>
<td>«destroy»</td>
</tr>
<tr>
<td>ref</td>
<td>«create»</td>
</tr>
</tbody>
</table>
Derivation of the Implementation (cont.)

Main Memory Objects

- **ludo : Game**
  - phase == "move"

- **theDie : Die**
  - value == 2

- **tom : Player**

- **albert : Player**

- **c1 : Counter**
  - 1: move()

- **pos3 : Position**
  - at

- **pos4 : Position**
  - next

- **pos5 : Position**
  - next

- **pos6 : Position**
  - at

Rule Diagram / Program

- **myPlayer**
  - counters

- **theDie**
  - counters

- **pos**
  - next
  - at

- **pos : Position**
  - at «destroy»

- **otherCounter : Counter**
  - counters

- **Counter : move 0 : Void**

- **value > 0**
  - value := value - 1

- **value <= 1**
Derivation of the Implementation (cont.2)

Main Memory Objects

- **ludo:**Game
  - phase == "move"

- **theDie:**Die
  - value == 1

- **tom:**Player
  - players
  - players current

- **albert:**Player
  - players
  - counters

- **c1:**Counter
  - 1: move()
  - counters

- **pos3:**Position
  - at
  - next

- **pos4:**Position
  - at
  - next

- **pos5:**Position
  - at
  - next

- **pos6:**Position
  - at

Rule Diagram / Program

- **myPlayer**
  - counters
  - at
  - value > 0
  - value := value - 1

- **theDie**
  - counters
  - at
  - value > 0
  - value := value - 1

- **Counter:**move 0: Void

- **pos:**Position
  - at "destroy"

- **this**
  - counters
  - myPlayer
  - otherPlayer

- **otherCounter:**Counter
Derivation of the Implementation (cont.3)

Main Memory Objects

- **ludo:Game**
  - phase == "move"

- **theDie:Die**
  - value == 0

- **tom:Player**
  - counters

- **albert:Player**
  - counters

- **c1:Counter**
  - 1: move()  
  - at

- **pos3:Position**
  - next
  - at

- **pos6:Position**
  - next

- **pos4:Position**
  - next

- **pos5:Position**
  - next

Rule Diagram / Program

- **myPlayer**
  - counters

- **theDie**
  - value > 0
  - value := value - 1

- **pos:Position**
  - at <destroy>

- **otherPlayer:Player**
  - counters

- **otherCounter:Counter**
  - counters
Derivation of the Implementation (cont.4)

Main Memory Objects

- **ludo:Game**
  - phase == "move"
- **theDie:Die**
  - value == 0
- **tom:Player**
- **albert:Player**
- **c1:Counter**
  - 1: move()
- **pos3:Position**
- **pos4:Position**
- **pos5:Position**
- **pos6:Position**

Rule Diagram / Program

- **Counter**
  - move 0: Void
  - value > 0
  - value := value - 1
- **theDie:Die**
- **myPlayer:Player**
- **otherPlayer:Player**
- **otherCounter:Counter**
- **pos**: next
- **pos**: at
- **pos**: next
- **pos**: destroy

© 2005 Albert Zündorf, University of Kassel
Derivation of the Implementation (cont.5)

Main Memory Objects

- ludo:Game
  - phase == "move"
  - die
  - players current
  - players

- theDie:Die
  - value == 0
  - die
  - counters

- tom:Player
  - counters

- albert:Player
  - counters

- c1:Counter
  - 1: move()
  - at

- pos3:Position
  - next

- pos6:Position
  - next

- pos4:Position
  - next

- pos5:Position
  - next

Rule Diagram / Program

- Counter:move 0: Void
  - counters
  - die

- this
  - counters
  - at
  - <create> at

- myPlayer:Player
  - value > 0
  - value := value - 1

- otherPlayer
  - <destroy>

- pos
  - at
  - <destroy>

- otherCounter
class Counter {
    public void move () { Position pos; . . .
        while (sdmSuccess) {
            try {
                sdmSuccess = false;
                pos = this.getAt ();
                JavaSDM.ensure (pos != null);
                next = pos.getNext ();
                JavaSDM.ensure (next != null);
                myPlayer = this.getOwner ();
                JavaSDM.ensure (myPlayer != null);
                theDie = myPlayer.getDie ();
                JavaSDM.ensure (theDie=value > 0);
                sdmSuccess = true;
                this.setAt (null);
                this.setAt (next);
                theDie.setV(theDie.getV() - 1); }
            catch (SDMException e) {} }
        // while
    }
}
Derivation of the Implementation (cont.7)
class Game {
    public void collectThrownCounters () {
        Iterator looserIter = this.iteratorOfPlayers();
        while (!sdmSuccess && looserIter.hasNext()) {
            try {
                sdmSuccess = false;
                looser = looserIter.next();
                lhome = looser.getHome();
                JavaSDM.ensure (lhome != null);
                countersIter = looser.iteratorOfCounters();
                while (!sdmSuccess && countersIter.hasNext()) {
                    try {
                        lostCounter = countersIter.next();
                        JavaSDM.ensure (lostCounter.getAt() == null);
                        sdmSuccess = true;
                        lostCounter.setAt (lhome);
                    } catch (SDMException e) {} // while
                } catch (SDMException e) {} // while
            } catch (SDMException e) {} // while
        } // while
    } // collectThrownCounters
} // Game
Derivation of the Implementation (cont.9)

- manual derivation of rule diagrams from stories
- brain required
- systematic guide lines provided e.g. in [SCESM04]
- automatic code generation [GraGra]
Summary

Story Driven Modeling

- model level analysis with story boards
- model level tests
- model level implementation with rule diagrams
- code generation
- model level testing / debugging

www.fujaba.de  zuendorf@uni-kassel.de
References


- [SCESM05] Leif Geiger, Albert Zündorf: *Story Driven Testing*; in proc. 4th International Workshop on Scenarios and State Machines: Models, Algorithms and Tools (SCESM'05) ICSE 2005 Workshop