FUJABA
Tool Suite

Meta-
CASE-
Tutorial

36. Story Driven Modeling with Graph Rewriting – A Practical Guide to Model-Driven Software Development

Version 13-1.0 09.01.18
Courtesy to Prof. Albert Zündorf, University of Kassel, Germany, Given in Dresden in 2005
http://www.se.eecs.uni-kassel.de/typo3/index.php?albert

Content

Fujaba Graph Rewriting Tool

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

Overview

Story Driven Modeling with Graph Rewriting:

Steps:

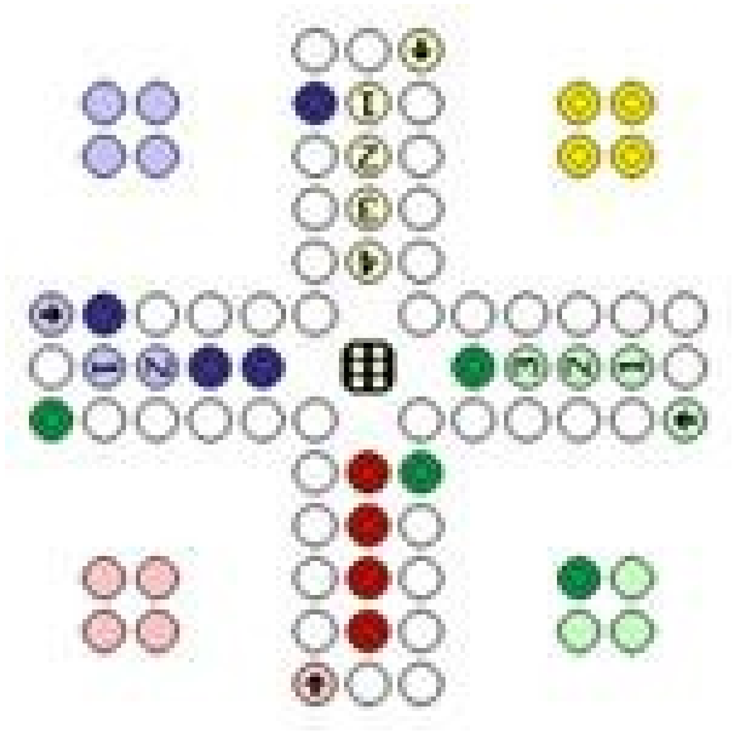
- Textual use case description
- Story Boarding (OOA)
  (Test specification)
- Class diagram derivation (OOD)
- Behavior derivation (Coding)
- Code generation
- Validation (Testing)

Features:

- Use Case Driven
- Model Driven
- Iterative
- Test Driven Development

42.2. The running example: Ludo

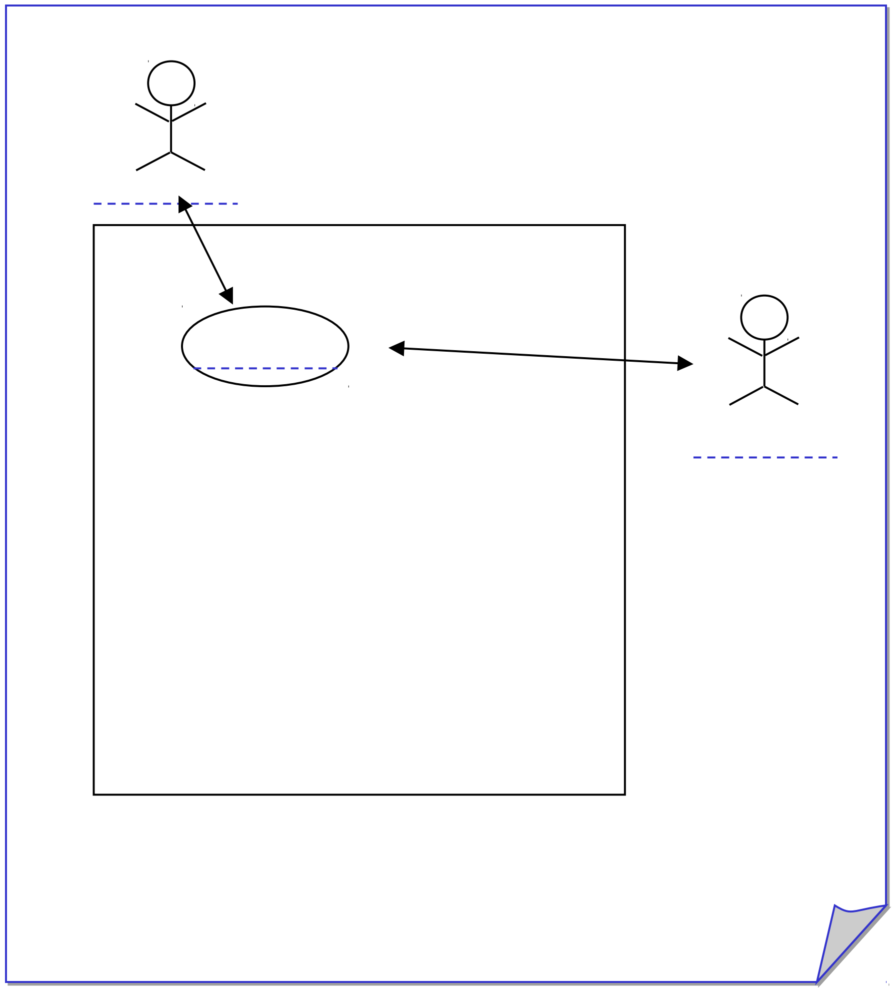
- Development of an interpreter for a language (here Ludo game)



42.3. Use case diagrams (Rpt.)

Requirements elicitation as usual:

- Use case diagrams for overview



Classic Use Case Description (cont.)

Textual scenario descriptions:

- focus on scenarios
- several scenarios per use case
- focus on one example situation at a time
- use concrete names

Use case _ _ _ _ _ _ _, _ _ _ _ _ _ _ :

Start situation: _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

Invocation: _ _ _ _ _ _ _ _ _ _ _ _ _

Step 1: _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

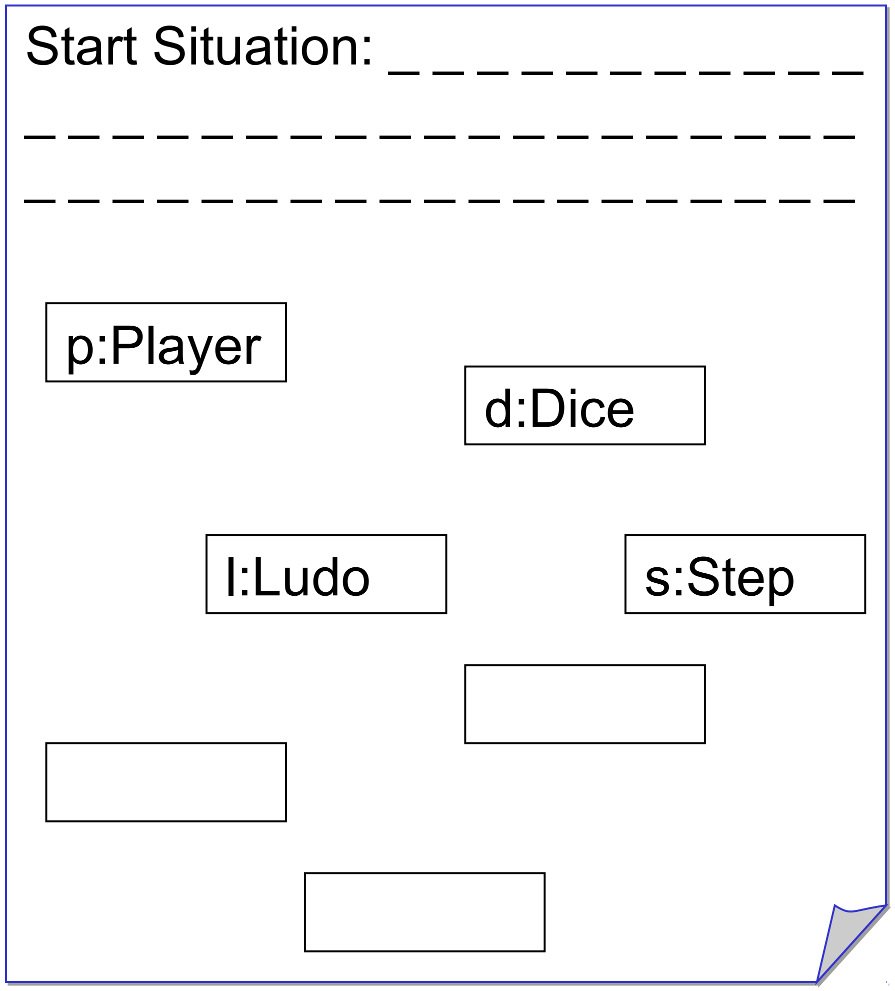
Step 2: _ _ _ _ _ _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

Result situation: _ _ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _

Story-Driven Modeling with Fujaba (SDM)

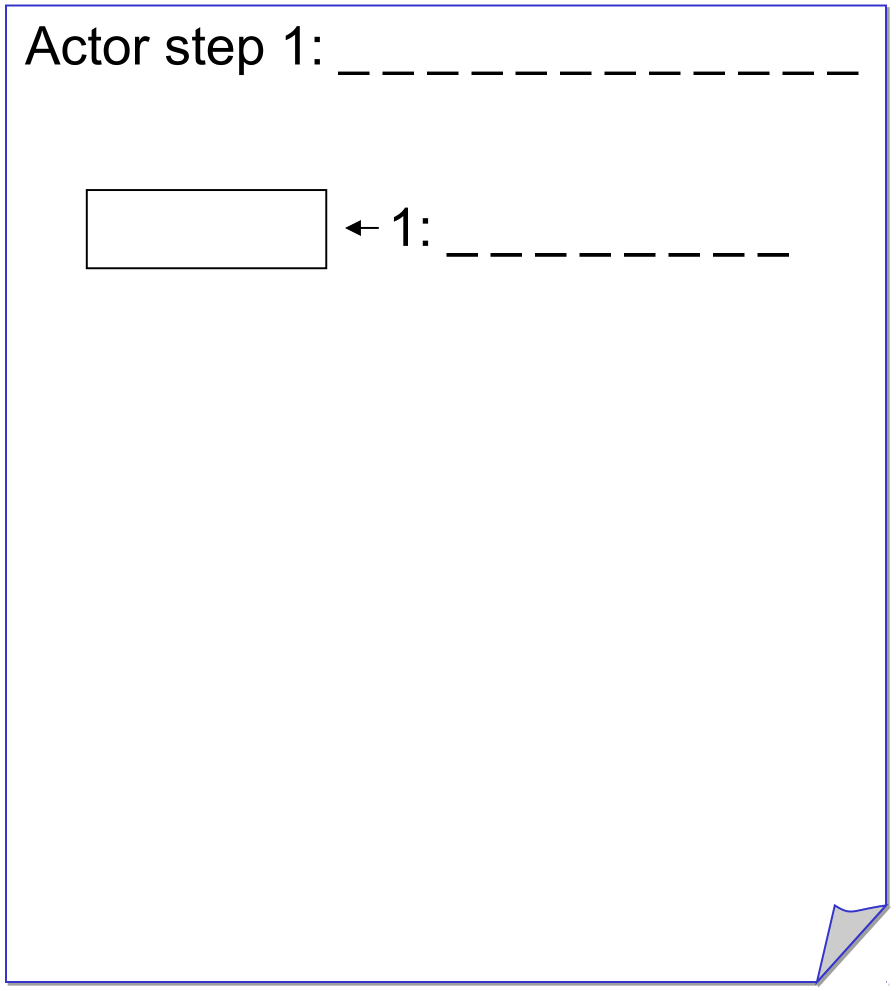
SDM approach is based on noun-verb-analysis:

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



42.4 Object-Oriented Scenario 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



Object-Oriented Scenario Analysis with Story Boards

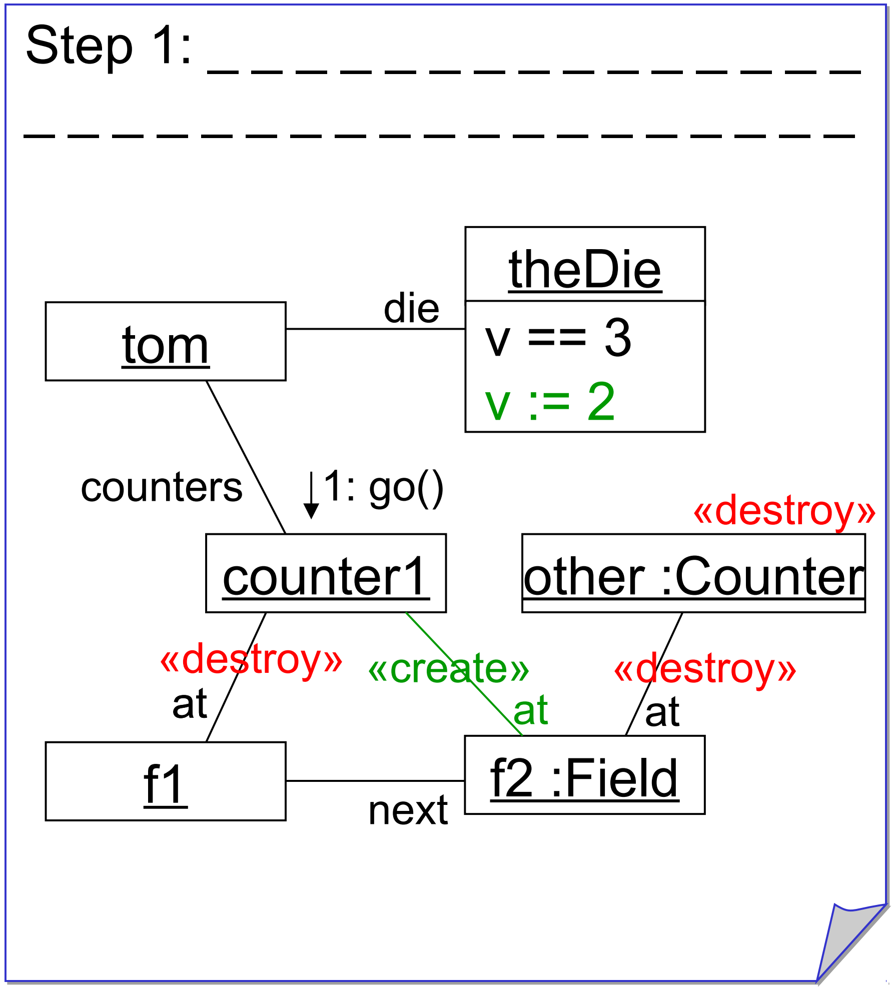
Relations in a use case are mapped to method calls

❍ 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 Scenario Analysis with Story Boards

- Change in scenarios are recorded by rewrite rules
- They outline method behavior in concrete scenarios

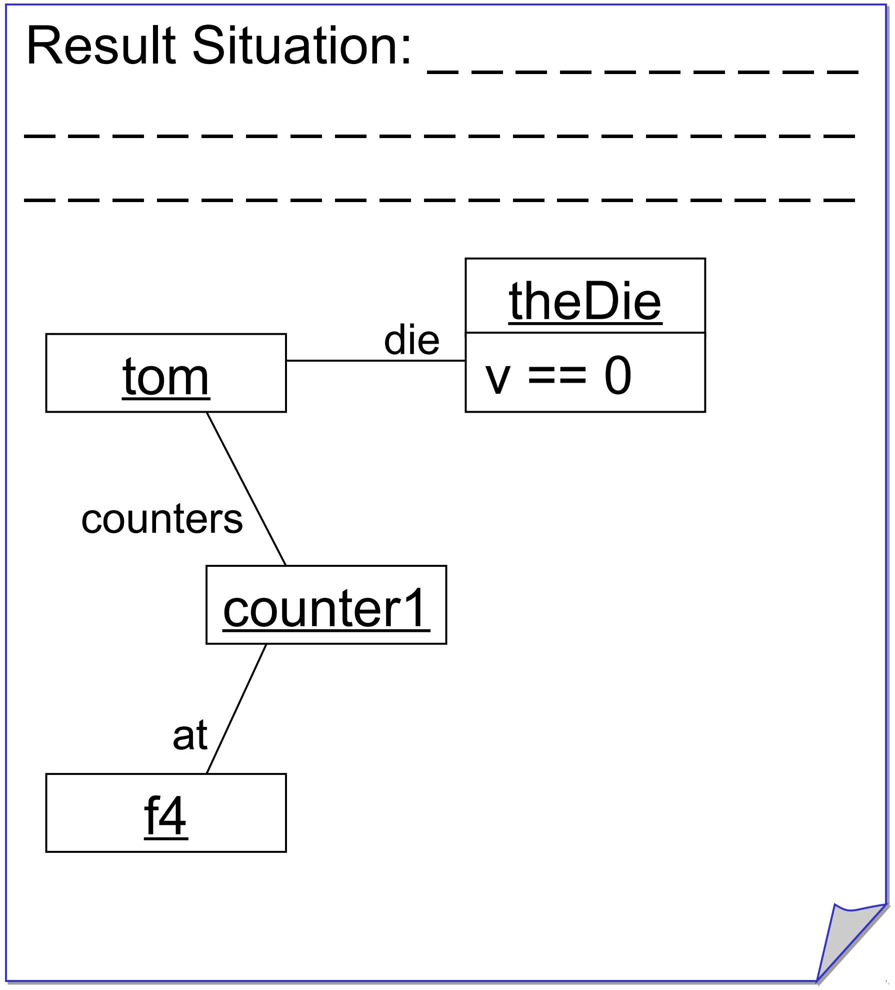
- <<create>> and <<destroy>> markers
- := attribute assignments
- recurring objects without class name
  first time on stage with class name (change of perspective)
- collaboration messages
- alternatively sequence diagrams



Object-Oriented Scenario Analysis with Story Boards

Result situation:

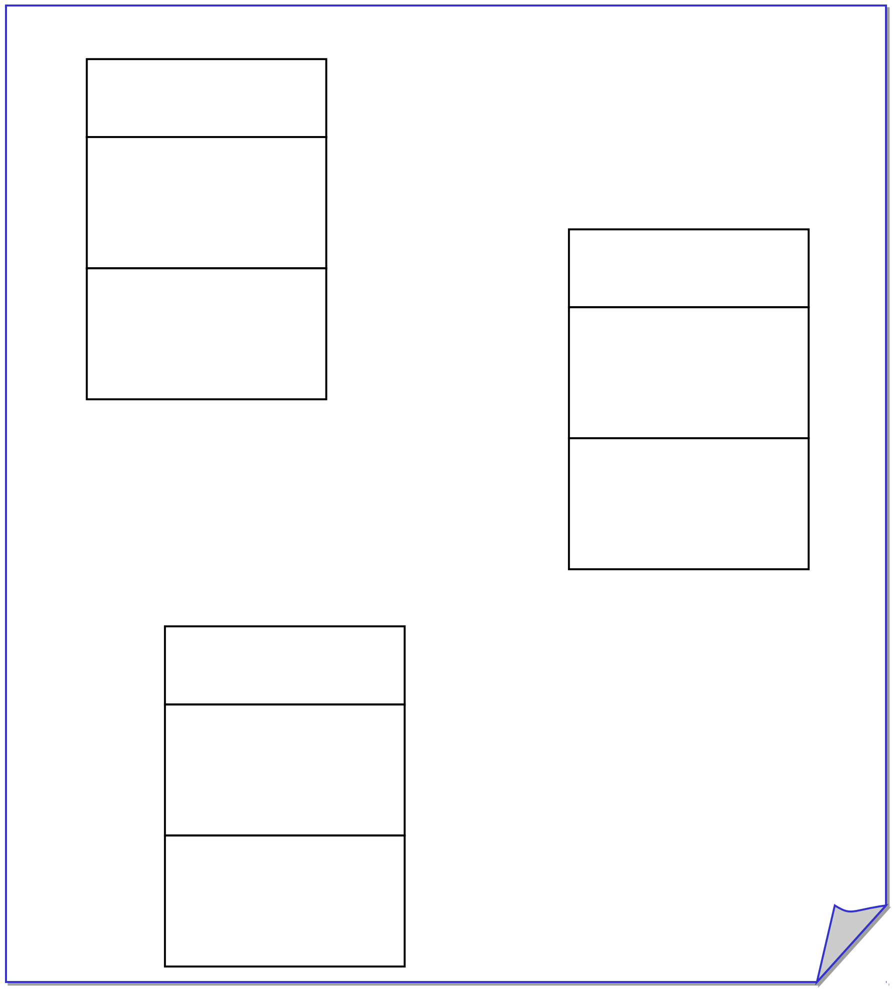
- models resulting object structure
- used for testing



Derivation of Class Diagrams from Scenarios

Collect the types from the story boards:

- Classes
- Associations
- Attribute declarations
- Method declarations



Derivation of Class Diagrams (cont.)

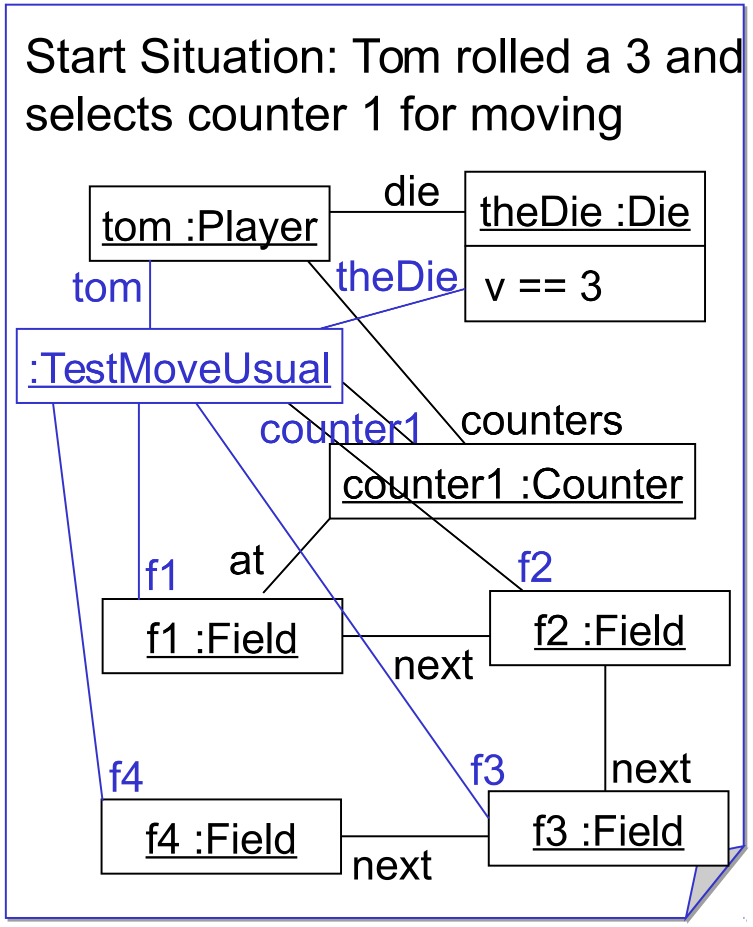
- Class diagram derivation is straight forward
- Semi-automatic tool support by Fujaba
- Intermediate story board step results in much better domain level class diagrams
- code generation for class diagrams
- *story boards are appropriate for the analysis and discussion of behavior*
- story boards also 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

Test Derivation (cont.)

- Scenarios → JUnit Tests, start situation → setup code and fixture

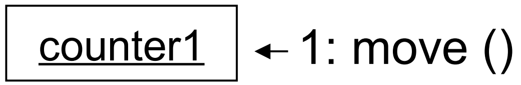


```
class TestMoveUsual implements TestCase {
    private Player tom;
    private Die theDie;
    private Counter counter1;
    ...
    void setUp () {
        tom = new Player ();
        theDie = new Die ();

        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

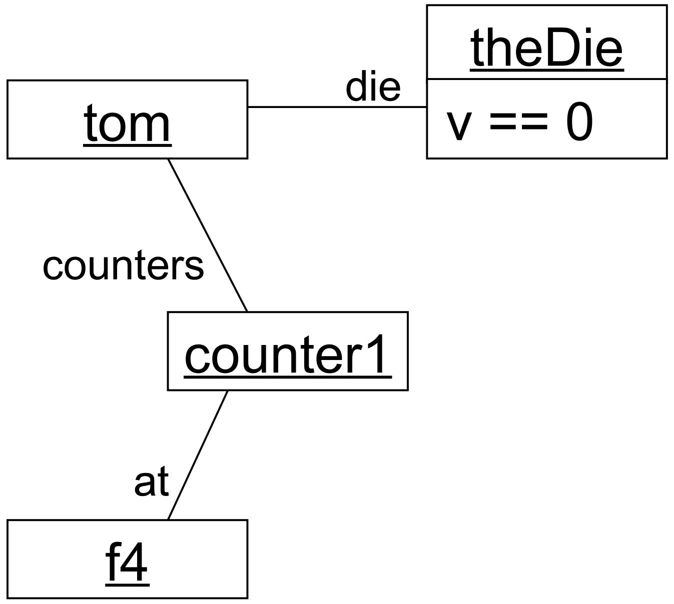


```
class TestMoveUsual implements TestCase
{
   . . .
   void testMoveUsual ()
   {
      this.counter1.move();
      . . .
```

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



```
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);
}
```

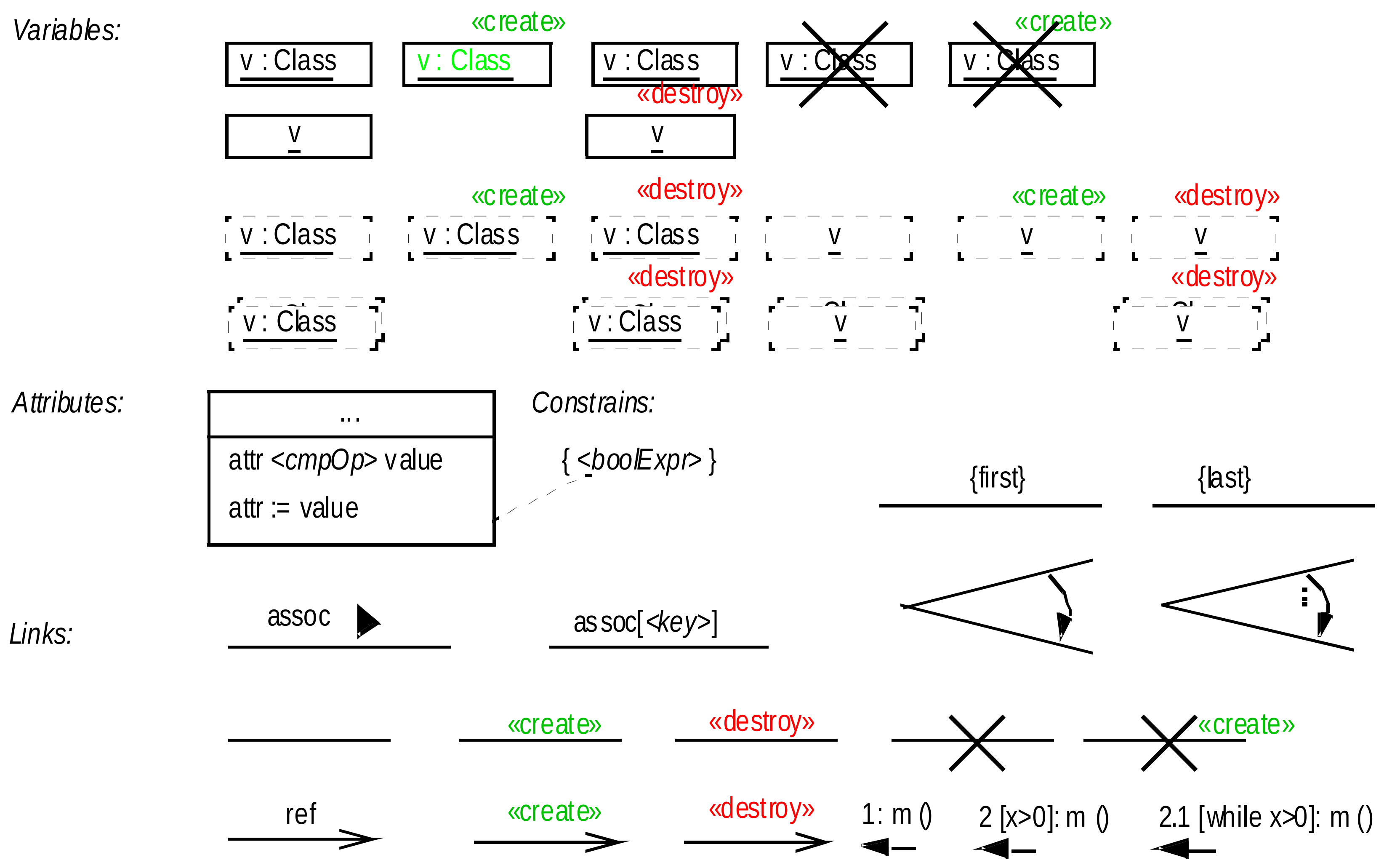
Test Derivation (cont.4)

- more complex result situations work, too (see later)
- start situation, invocation, result situation → JUnit tests
- steps may be exploited, too, cf. [SCESM05]
- analysis scenarios ←→ tests
- test driven software development

42.6 Derivation of the Implementation

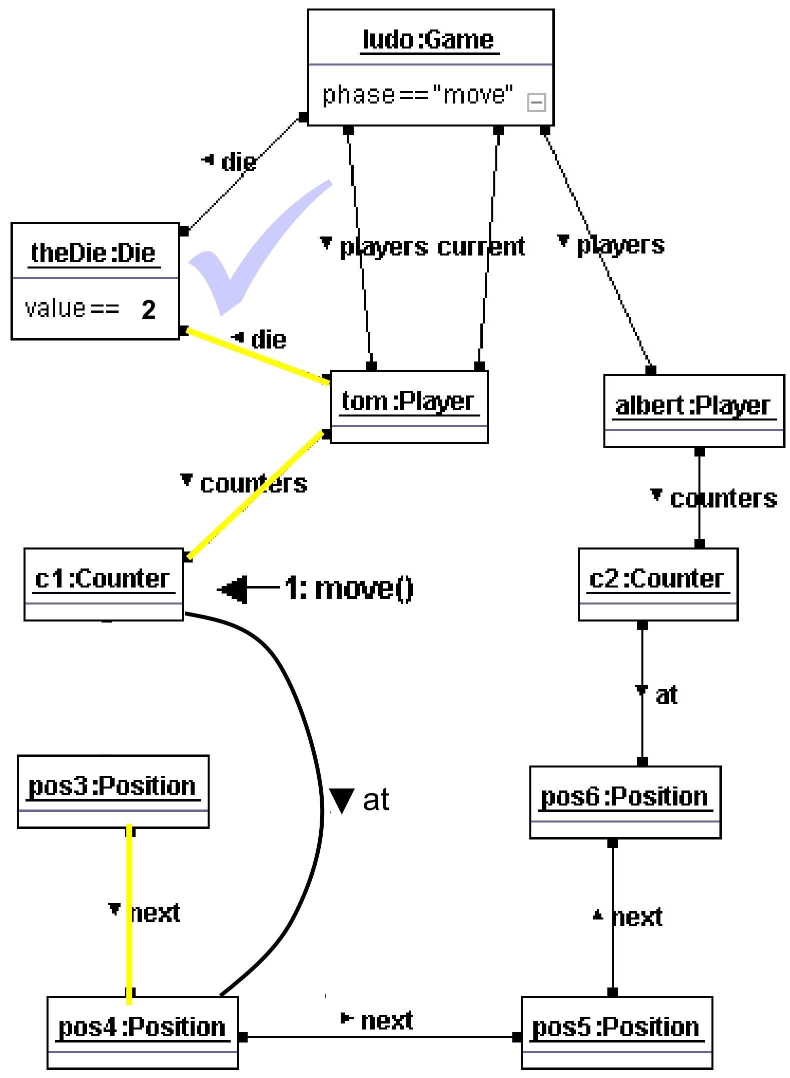
- combine story boards to rule diagrams [SCESM04]
- assign execution semantics
- code generation

Story Pattern Elements:

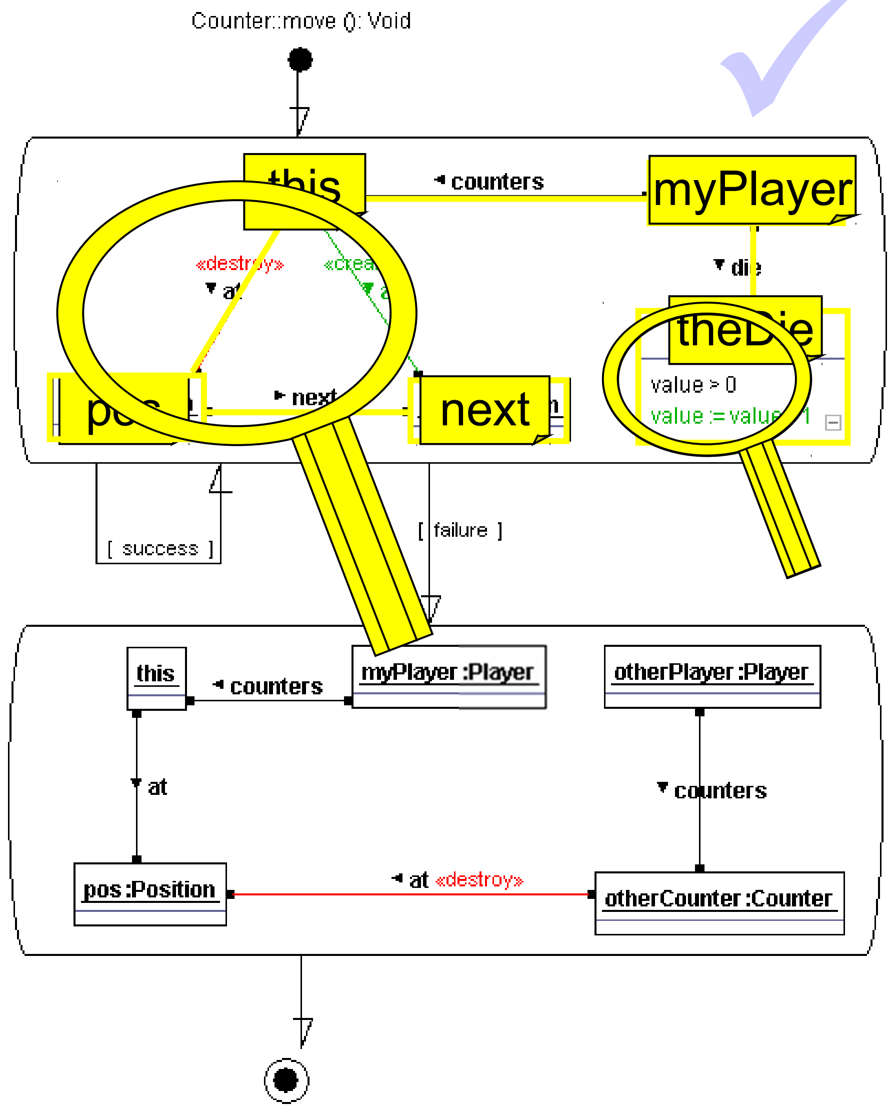


Derivation of the Implementation (cont.)

Main Memory Objects

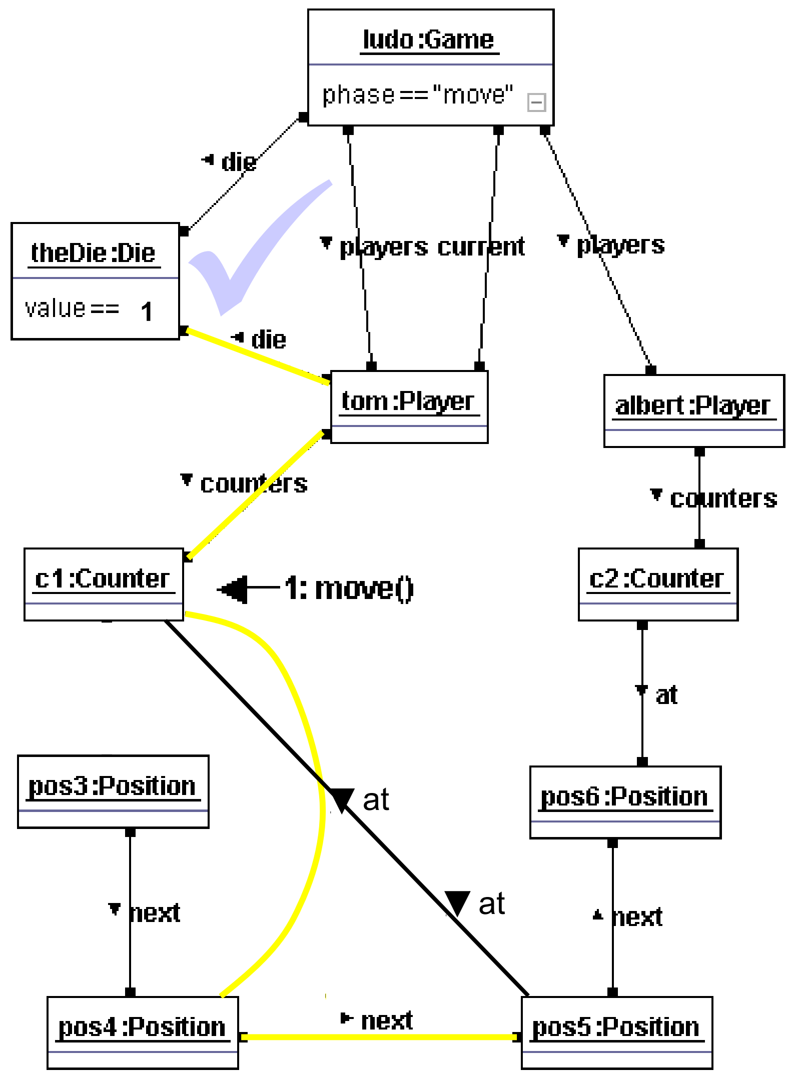


Rule Diagram / Program

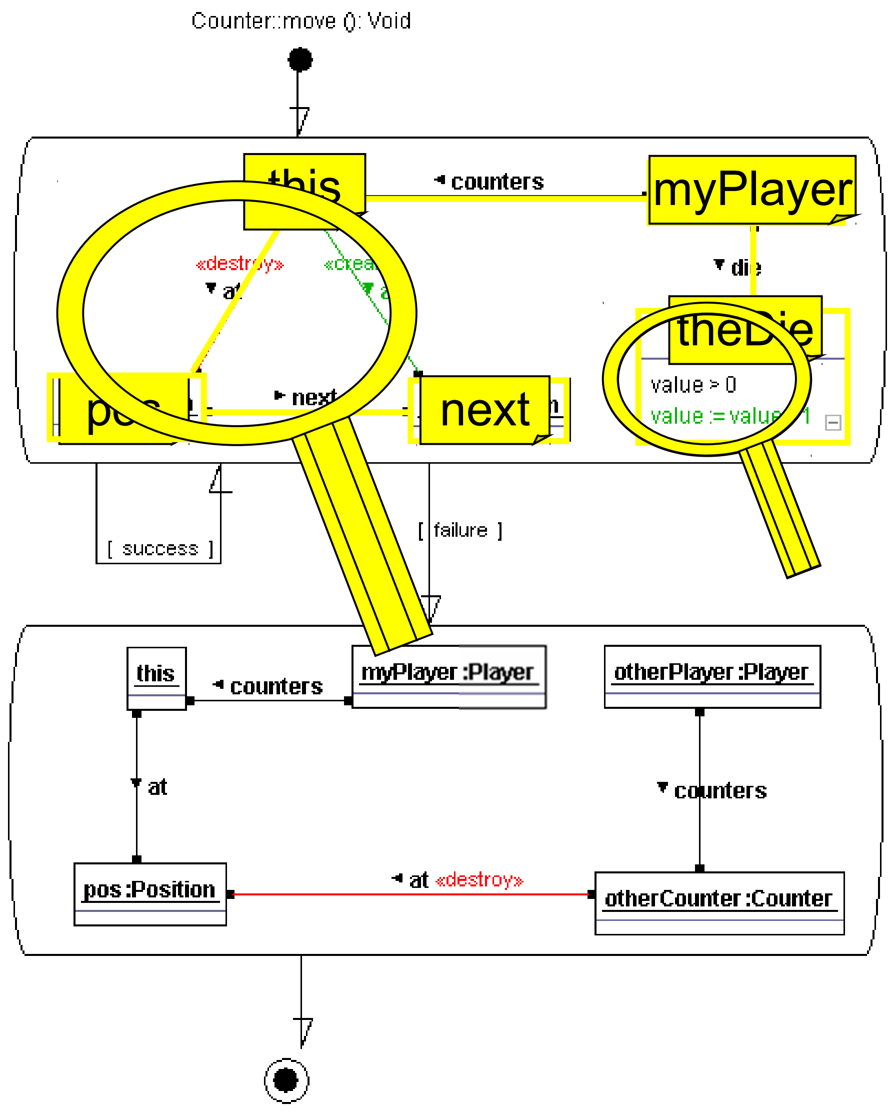


Derivation of the Implementation (cont.2)

Main Memory Objects

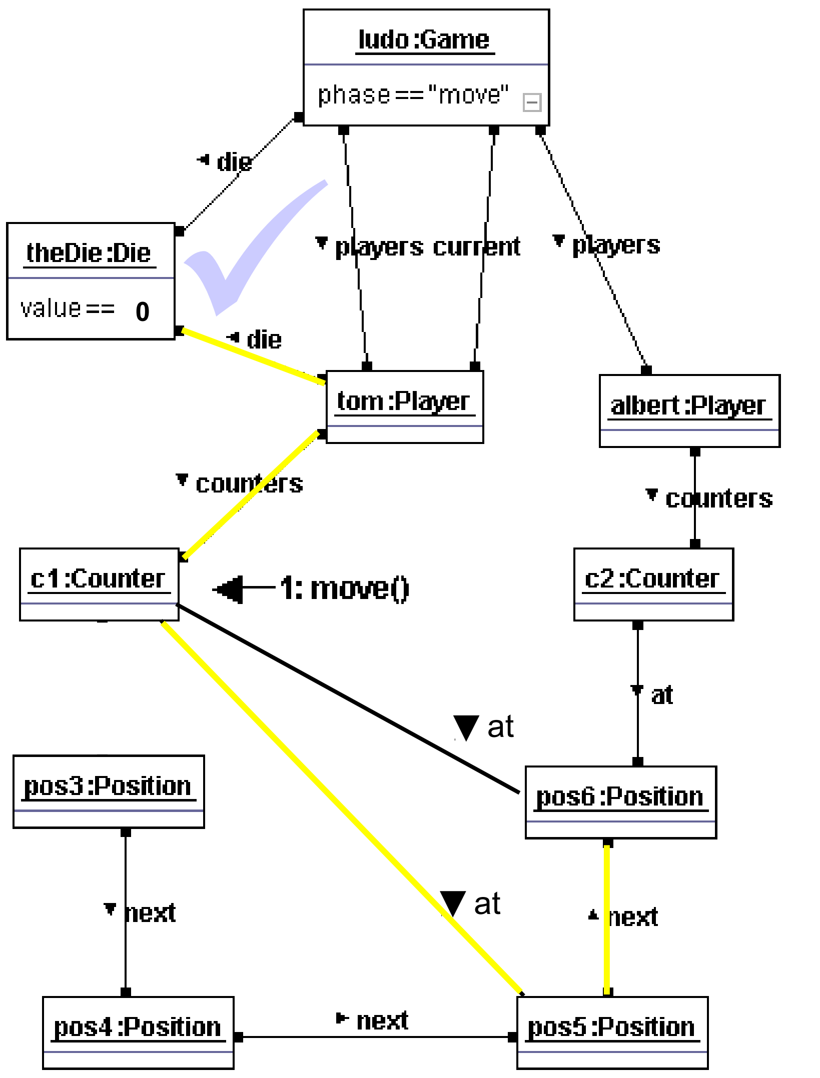


Rule Diagram / Program

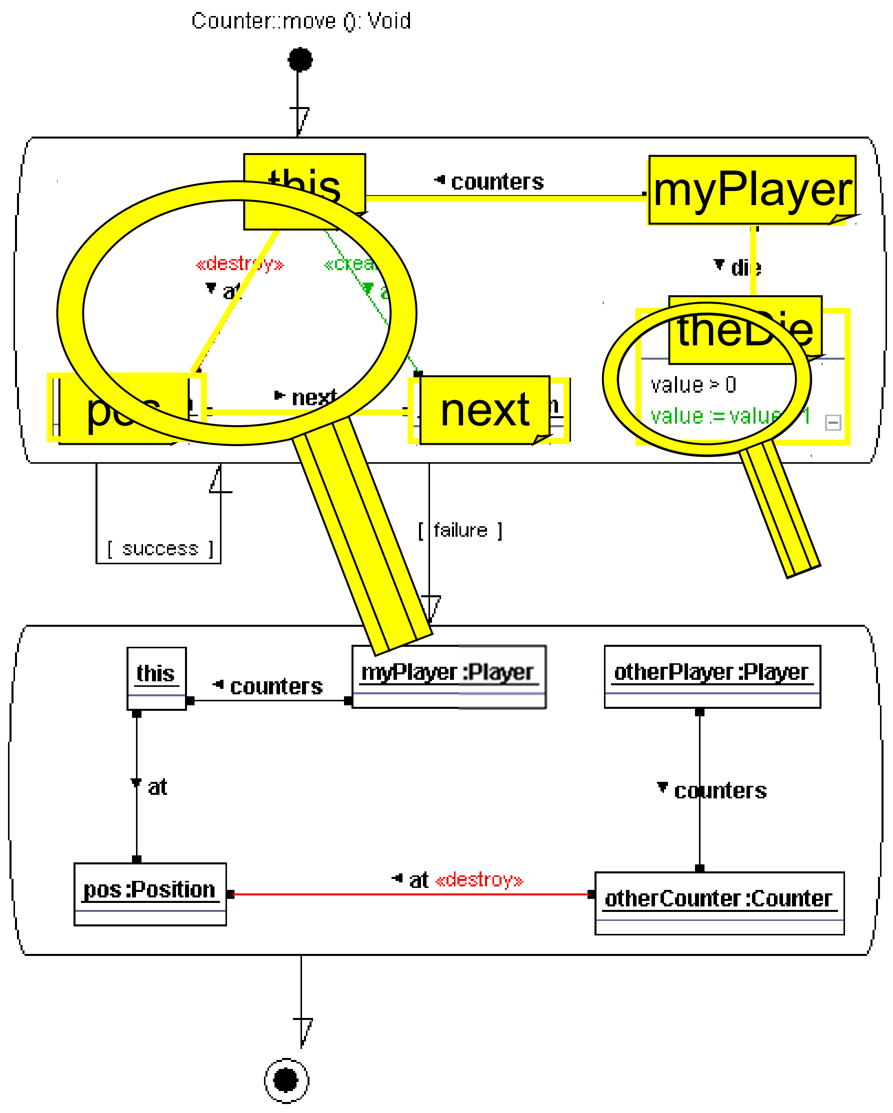


Derivation of the Implementation (cont.3)

Main Memory Objects

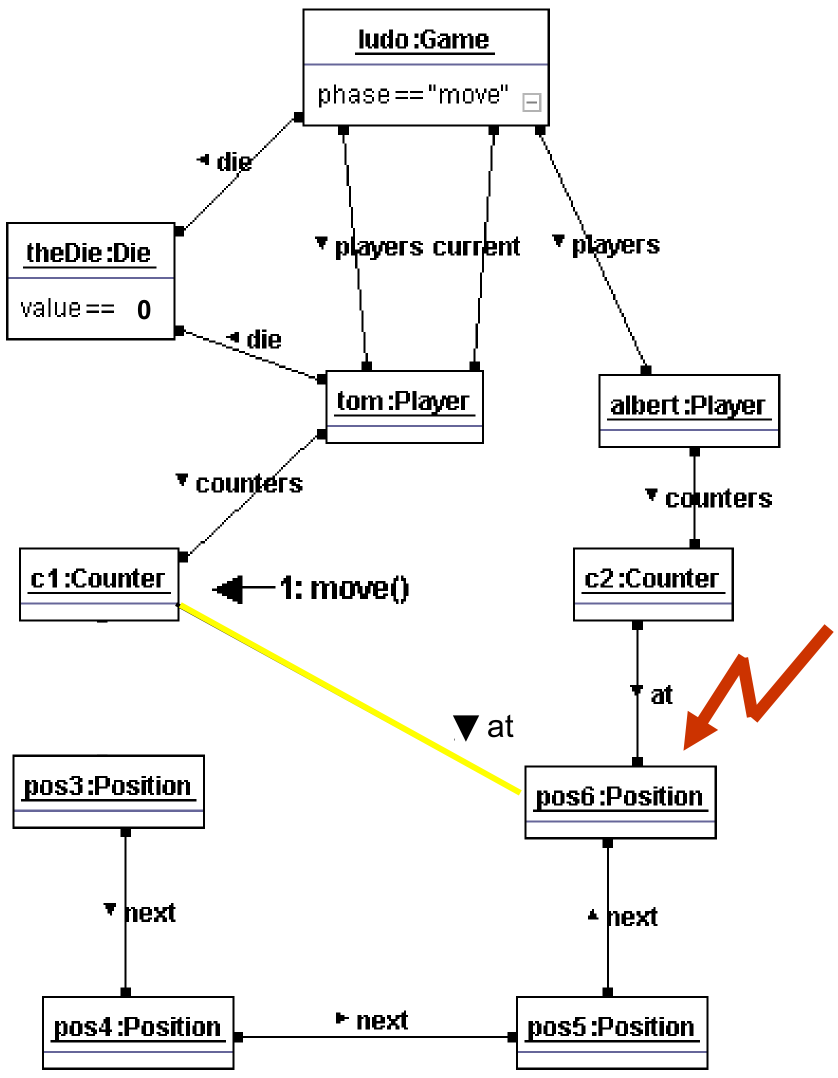


Rule Diagram / Program

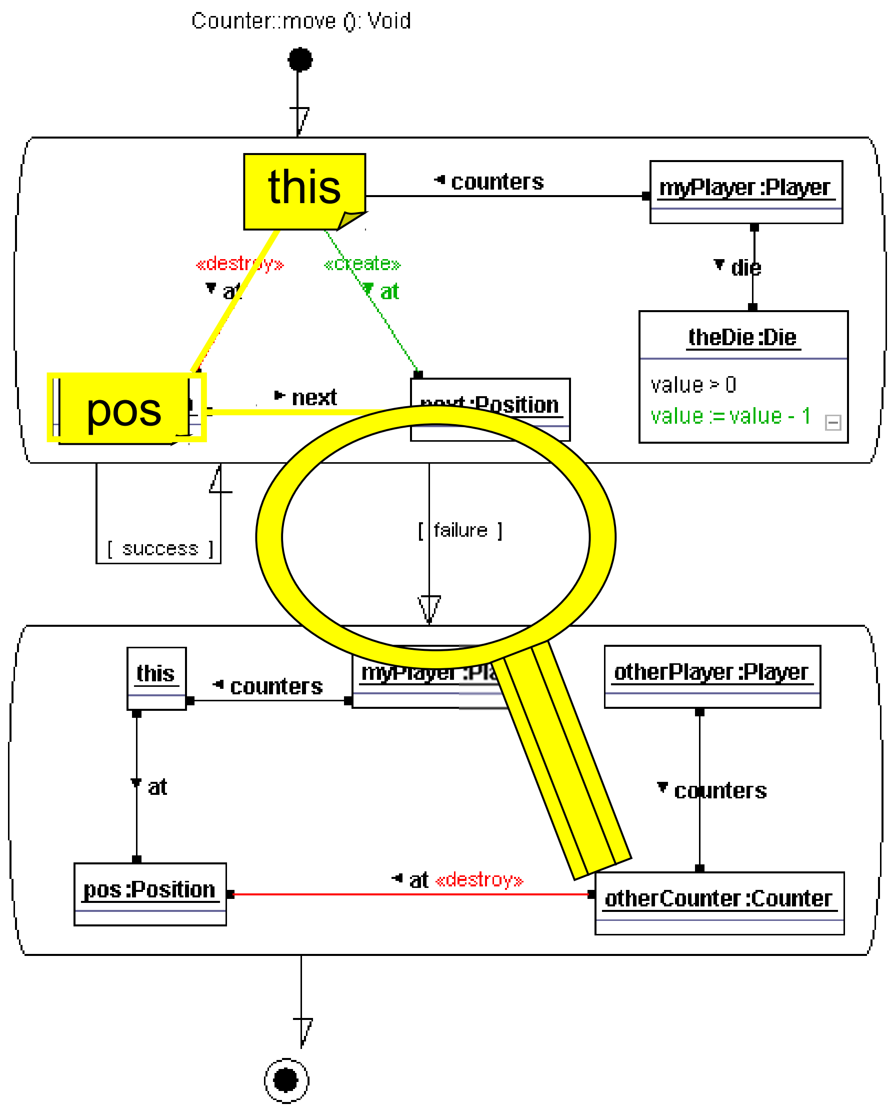


Derivation of the Implementation (cont.4)

Main Memory Objects

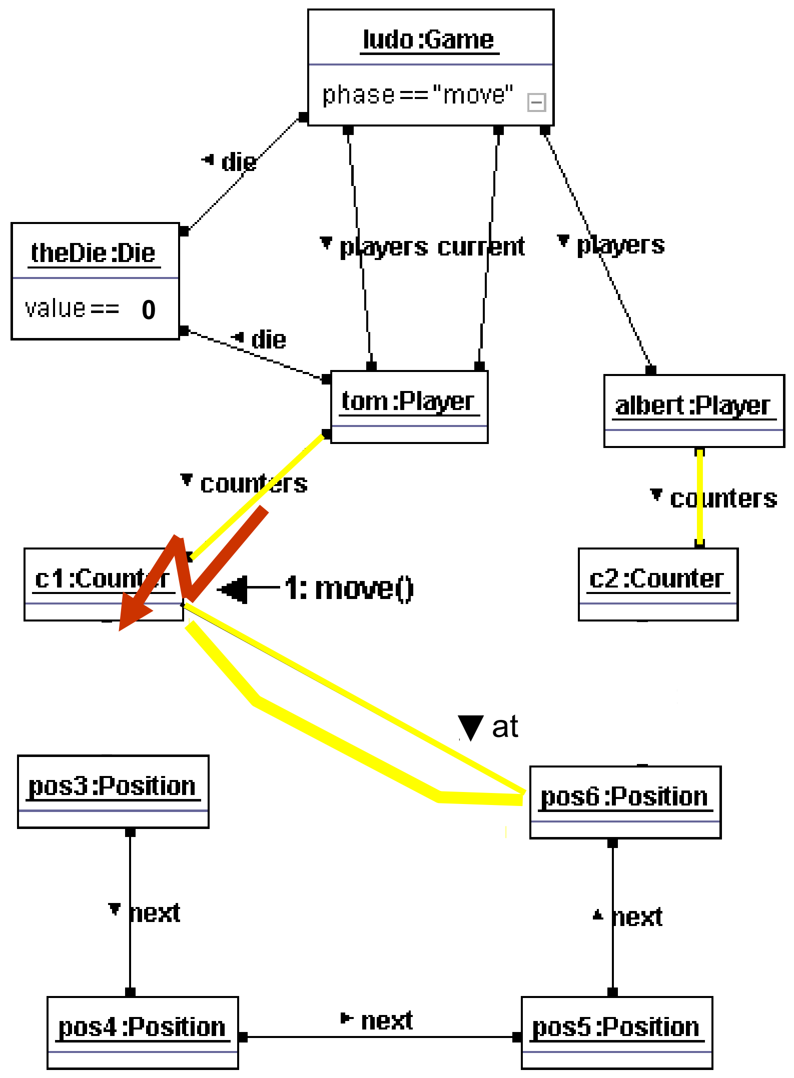


Rule Diagram / Program

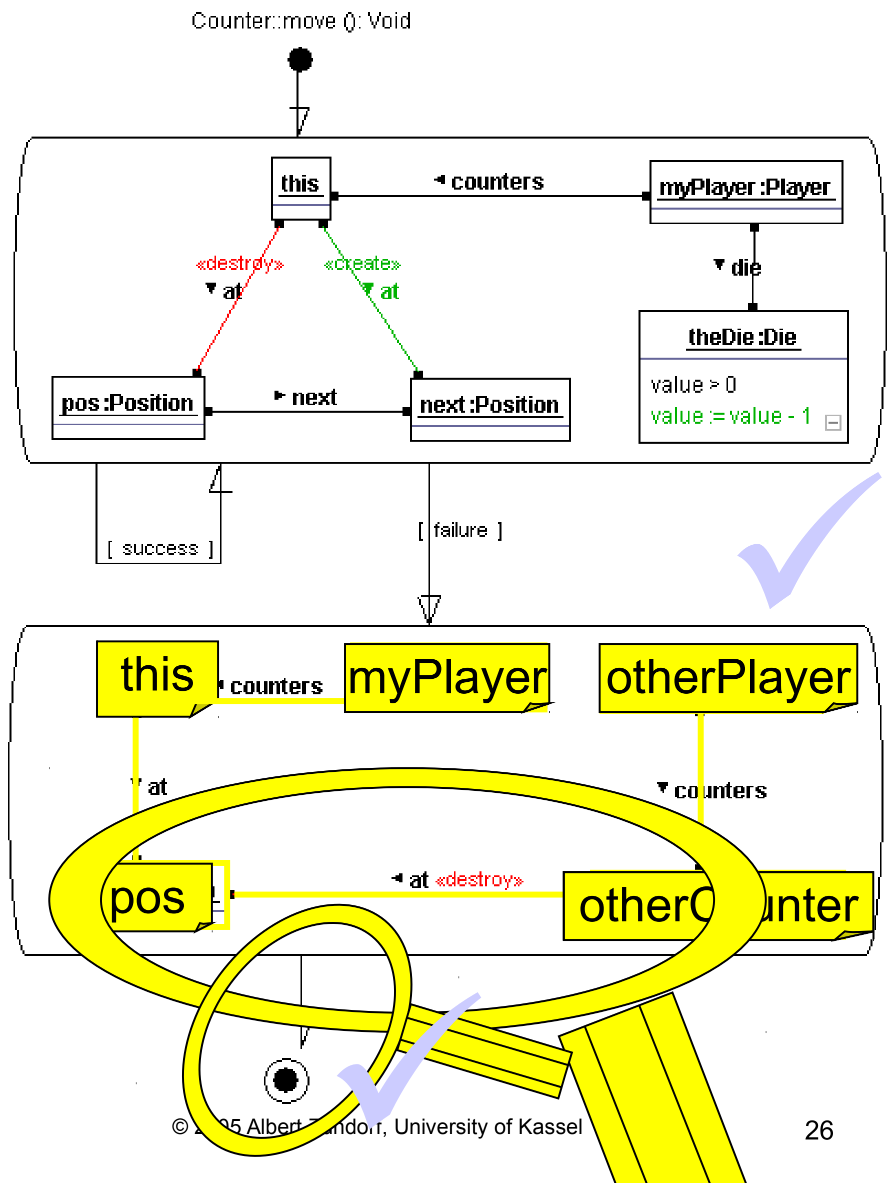


Derivation of the Implementation (cont.5)

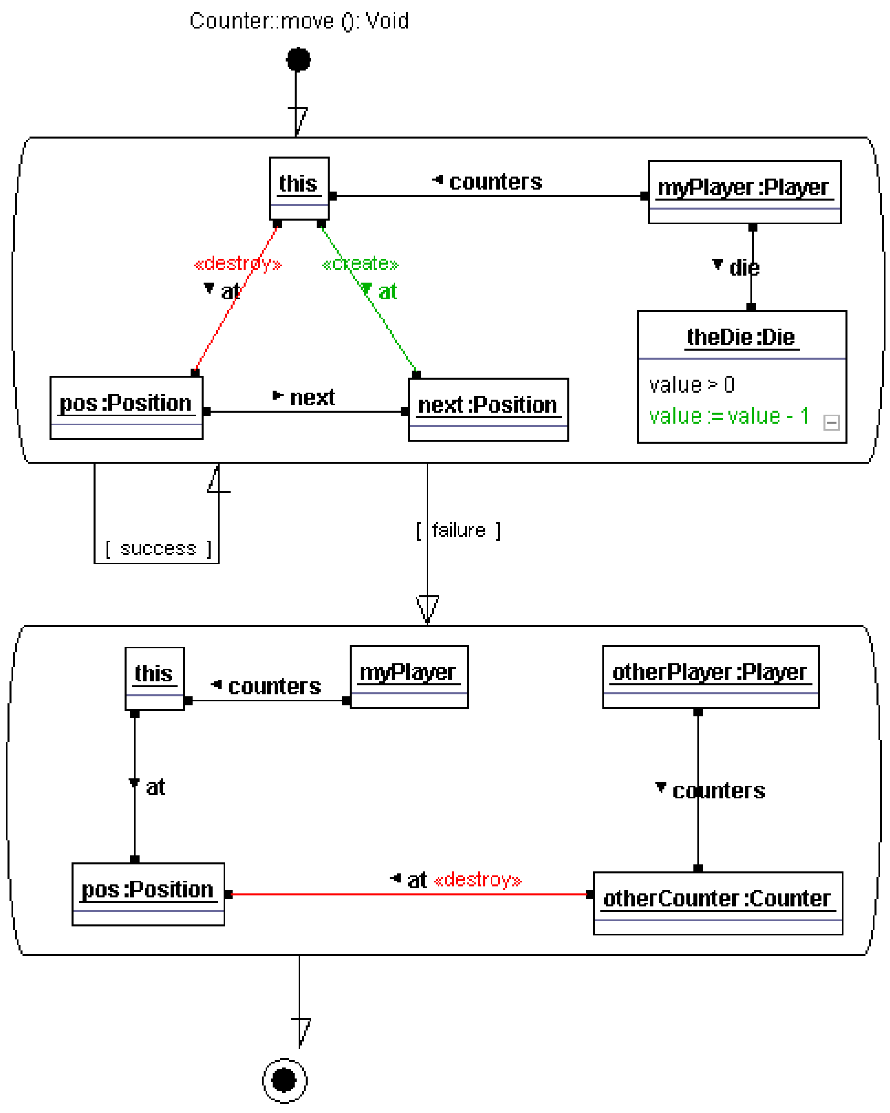
Main Memory Objects



Rule Diagram / Program

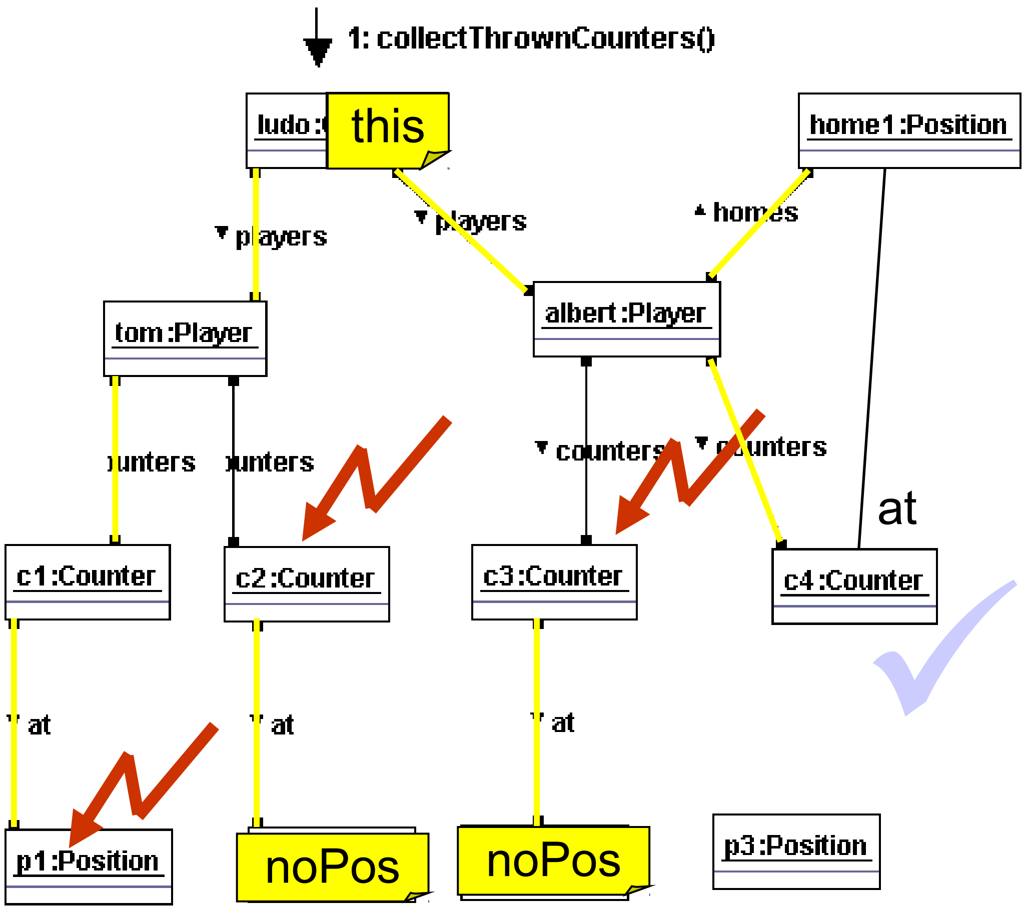


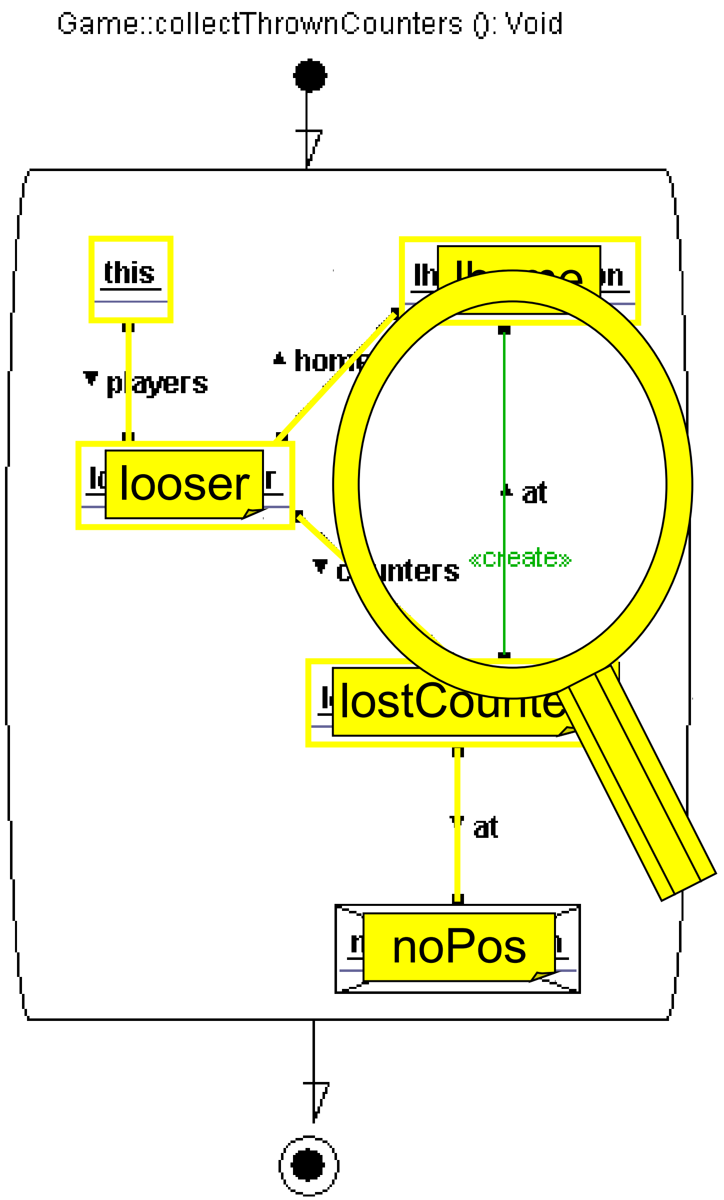
Derivation of the Implementation (cont.6)



```
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 != null);
   JavaSDM.ensure (theDie.getV() > 0);
   sdmSuccess = true;
   this.setAt (null);
   this.setAt (next);
   theDie.setV(theDie.getV() - 1); }
   catch (SDMException e) {}
} // while
```

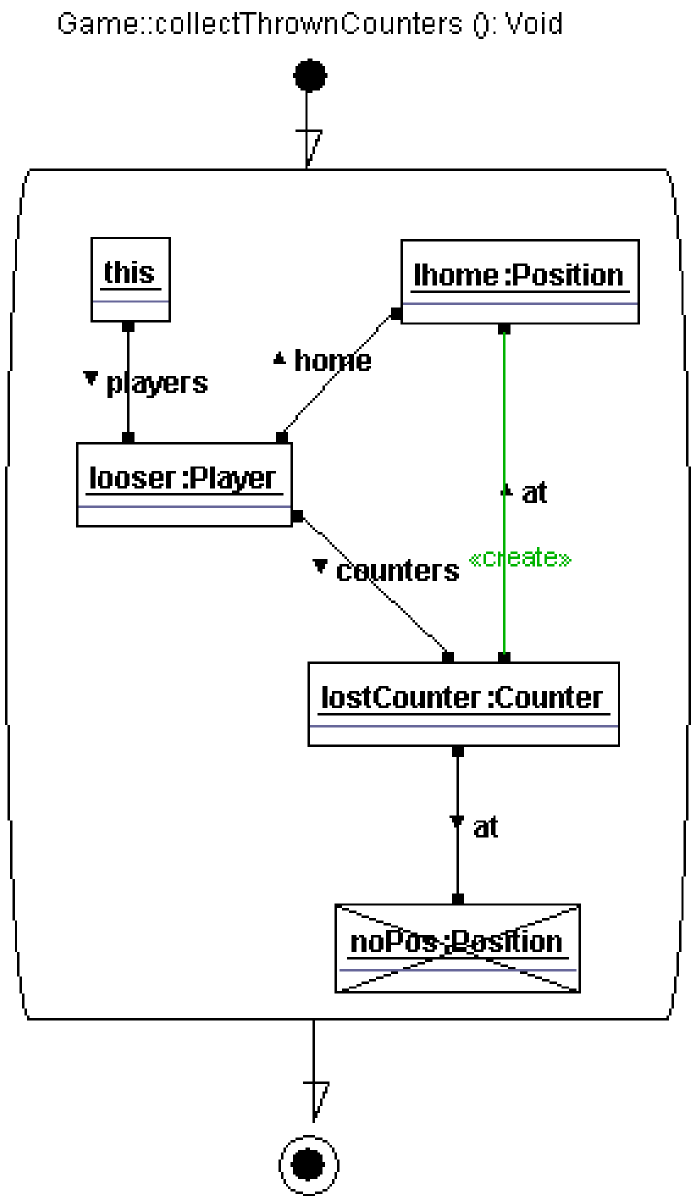
Derivation of the Implementation (cont.7)





Derivation of the Implementation (cont.8)

```
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
```



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

- [SCESM04] I. Diethelm, L. Geiger, A. Zündorf: *Systematic Story Driven Modeling, a case study*; Workshop on Scenarios and state machines: models, algorithms, and tools; ICSE 2004, Edinburgh, Scottland, May 24 – 28 (2004).
- [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
- [GraGra] T.Fischer, J.Niere, L.Torunski, A.Zündorf: *Story Diagrams: A new Graph Grammar Language based in the Unified Modeling Language*; in Proc. of TAGT '98 - 6th International Workshop on Theory and Application of Graph Transformation. Technical Report tr-ri-98-201, University of Paderborn; (1999)