Formal Object-oriented Software Development

Priv.-Doz. Dr. Rolf Hennicker

April 22, 2002
Chapter 1: Software Development and Formal Specification

Goals

• to understand the basic principles of software development
• to understand the role of formal methods in software development
• to understand the basic principles of object-oriented systems and their formal specification
1.1 Introduction

Software Engineering
deals with the technical and organisational aspects of the
development and maintenance of large software systems.

State of the Art
• natural language documentations
• diagrammatic modeling languages (e.g. UML)
• CASE Tools (Computer Aided Software Engineering)
• formal methods (for safety critical systems)
Formal Methods

• based on mathematics (set theory, algebra, math. logics)
• advantages:
  ◦ unique interpretation of syntactic constructs
  ◦ verification of properties (of specifications, models, programs)
  ◦ verification of the correctness of development steps
• difficulties:
  ◦ knowledge of formal notations and their meaning
  ◦ additional development costs
1.2 Software Development Process

Process Models

- waterfall model

- iterative model, V-model, spiral model, .......

Formal software development uses additionally *formal specifications* in the different phases. Verification of the correctness of a development step is only possible on the basis of formal specifications.
1.3 Formal Object-oriented Software Development

Object-oriented System
consists of a set of objects which communicate by exchanging messages. If an object receives a message it performs an operation (method) which possibly changes the (local) state of the object.

Class
describes a set of objects with common properties (attributes) and operations. The actual value of the attributes of an object (at a certain time) determines the state of the object.
Process Model (simplified)

<table>
<thead>
<tr>
<th>Requirements Analysis</th>
<th>Design Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>• identification of use cases and actors</td>
<td>• refinement of the static model</td>
</tr>
<tr>
<td>• UML use case model</td>
<td>• component-based system</td>
</tr>
<tr>
<td>• description of scenarios</td>
<td>architecture</td>
</tr>
<tr>
<td></td>
<td>• formal annotations</td>
</tr>
<tr>
<td>(formal) realization relation</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Requirements Specification</th>
<th>Implementation</th>
</tr>
</thead>
<tbody>
<tr>
<td>• static model (class diagram)</td>
<td>• realization of the design in an</td>
</tr>
<tr>
<td>• dynamic model (sequence diagrams, state diagrams)</td>
<td>object-oriented programming</td>
</tr>
<tr>
<td>• formal annotations (e.g. invariants)</td>
<td>language</td>
</tr>
</tbody>
</table>

(formal) realization relation
Proving the Correctness of the Implementation

a) Verification of the implementation against the requirements specification (post mortem) OR

b) Verification of each realization step (verification conditions)

Remarks

• Testing can only show the existence of errors. Verification can show the absence of errors.
• The adequateness of a (formal) specification w.r.t. the desires of the user can not be verified.
OCL: Object Constraint Language

- is a formal language to express (additional) constraints on UML models
- is originally developed at IBM (1995)
- should be easy to use (without deep mathematical knowledge)

Applications of OCL

- for constraining the UML meta model (i.e. for language definitions)
- for specifying invariants of classes
- for specifying pre- and post-conditions of operations
- for expressing guards in state diagrams
1.4 Example: Points and Shapes

Requirements

• Develop a system for the representation and manipulation of points and shapes
• A shape consists of a set of points
• A point is determined by its coordinates
• A point can be added or removed from a shape
• Points and shapes can be moved
• It is possible to decide whether a point belongs to a shape or not
A first static model
Static model for Points and Shapes

```
<table>
<thead>
<tr>
<th>System</th>
</tr>
</thead>
<tbody>
<tr>
<td>createPoint(x: Real, y: Real): Point</td>
</tr>
<tr>
<td>createShape(): Shape</td>
</tr>
<tr>
<td>addPointToShape(p: Point, s: Shape)</td>
</tr>
<tr>
<td>removePointFromShape(p: Point, s: Shape)</td>
</tr>
<tr>
<td>movePoint(p: Point, mx: Real, my: Real)</td>
</tr>
<tr>
<td>moveShape(s: Shape, mx: Real, my: Real)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Point</th>
<th>0..1</th>
<th>Shape</th>
</tr>
</thead>
<tbody>
<tr>
<td>xx: Real</td>
<td></td>
<td></td>
</tr>
<tr>
<td>yy: Real</td>
<td></td>
<td></td>
</tr>
<tr>
<td>move(mx: Real, my: Real)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>setOwner(s: Shape)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>unsetOwner()</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Point</th>
<th>0..1</th>
<th>Shape</th>
</tr>
</thead>
<tbody>
<tr>
<td>addPoint(p: Point)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>removePoint(p: Point)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>move(mx: Real, my: Real)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>belongsTo(p: Point): Boolean {query}</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

pointSet * → Point

figures * → Shape
1.4 Example: Points and Shapes

Static Model with Constraints

System

createPoint(x:Real, y:Real):Point
createShape():Shape
addPointToShape(p:Point, s:Shape)
removePointFromShape(p:Point, s:Shape)
movPoint(p:Point, mx:Real, my:Real)
movShape(s:Shape, mx:Real, my:Real)

inv: pointSet -> forAll(p
p.owner.points->includes(p)) and
figures->forAll(s|
s.points->forAll(p|p.owner=s))

pre: pointSet->includes(p) and
figures->includes(s) and
p.owner=null
post: s.points=s.points@pre->
including(p) and p.owner=s

* pointSet

Point

xx:Real
yy:Real

move(mx:Real,my:Real)
setOwner(s:Shape)
unsetOwner()

post: xx=xx@pre+mx and
yy=yy@pre+my

* figures

Shape

R. Hennicker: 1 Software Development and Formal Specification, April 22, 2002
1.5 Summary

- Software development uses process models
- Formal methods can be integrated in all phases of software development
- Correctness of an implementation can only be proved in the presence of formal specifications
- An object-oriented system consists of a set of communicating objects
- OCL is a formal language for specifying properties of object-oriented systems like invariants, pre- and post-conditions