Contents

- Overview of KeY
- **UML and its semantics**
- Introduction to OCL
- Specifying requirements with OCL
- Modelling of Systems with Formal Semantics
- Propositional & First-order logic, sequent calculus
- OCL to Logic, horizontal proof obligations, using KeY
- Dynamic logic, proving program correctness
- Java Card DL
- Vertical proof obligations, using KeY
- Wrap-up, trends
Building Models

Real World
Rough Idea of System

UML Diagrams

OCL-Constraints

Java - Implementation

Model

Increasing Formalisation

Verification
Unified Modeling Language

Unified: end to many similar approaches.

Booch, Rumbaugh, Jacobsson

Standardised by OMG (now version 2.0 in finalisation)
Unified Modeling Language

Unified: end to many similar approaches.
Booch, Rumbaugh, Jacobsson
Standardised by OMG (now version 2.0 in finalisation)

Modeling: main (creative) process of software development
Trend in SWE: emphasis on model, MDA/MDE
Code abstraction, formal model
Unified Modeling Language

Unified: end to many similar approaches.
Booch, Rumbaugh, Jacobsson
Standardised by OMG (now version 2.0 in finalisation)

Modeling: main (creative) process of software development
Trend in SWE: emphasis on model, MDA/MDE
Code abstraction, formal model

Language: Provides notation, no method, no process
Graphical, collection of different diagram types
UML Diagrams

- Structural Diagrams
- Behavioural Diagrams
UML Diagrams

- **Structural Diagrams**
  - Class Diagrams
  - Component Diagrams
  - Composite Structure Diagrams
  - Object Diagrams
  - Deployment Diagrams
  - Package Diagrams

- **Behavioural Diagrams**
UML Diagrams

- **Structural Diagrams**
  - Class Diagrams
  - Component Diagrams
  - Composite Structure Diagrams
  - Object Diagrams
  - Deployment Diagrams
  - Package Diagrams

- **Behavioural Diagrams**
UML Diagrams

Structural Diagrams
- Class Diagrams
- Component Diagrams
- Composite Structure Diagrams
- Object Diagrams
- Deployment Diagrams
- Package Diagrams

Behavioural Diagrams
UML Diagrams

- **Structural Diagrams**
  - Class Diagrams
  - Component Diagrams
  - Composite Structure Diagrams
  - Object Diagrams
  - Deployment Diagrams
  - Package Diagrams

- **Behavioural Diagrams**
  - Activity Diagrams
  - Interaction Diagrams
    - Sequence Diagrams
    - Collaboration Diagrams
  - Use Case Diagrams
  - State Machine Diagrams
UML Diagrams

Structural Diagrams
- Class Diagrams
- Component Diagrams
- Composite Structure Diagrams
- Object Diagrams
- Deployment Diagrams
- Package Diagrams

Behavioural Diagrams
- Activity Diagrams
- Interaction Diagrams
  - Sequence Diagrams
  - Collaboration Diagrams
- Use Case Diagrams
- State Machine Diagrams
UML Diagrams

- **Structural Diagrams**
  - Class Diagrams
  - Component Diagrams
  - Composite Structure Diagrams
  - Object Diagrams
  - Deployment Diagrams
  - Package Diagrams

- **Behavioural Diagrams**
  - Activity Diagrams
  - Interaction Diagrams
    - Sequence Diagrams
    - Collaboration Diagrams
  - Use Case Diagrams
  - State Machine Diagrams
Class Diagrams

Model static design view, define vocabulary (signature)

Class

Collection of similar objects in a system

<table>
<thead>
<tr>
<th>Attributes</th>
<th>Operations/Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>name: String</td>
<td>startWork(t:Time)</td>
</tr>
<tr>
<td>e-mail: String</td>
<td></td>
</tr>
</tbody>
</table>
Class Diagrams

Model static design view, define vocabulary (signature)

**Class**
Collection of similar objects in a system

**Attributes**
Operations/Methods

**Association**
Relation between classes
Relates pairs of class instances

- **Person**
  - name: String
  - e-mail: String
  - startWork(t: Time)

- **Car**
  - 0..* repairs
  - mechanic

- **Person** → **Car**
  - 1 mechanic
  - 0..* repairs
Class Diagrams

Model static design view, define vocabulary (signature)

Class
Collection of similar objects in a system

Attributes Operations/Methods

Association
Relation between classes
Relates pairs of class instances

Generalisation/Inheritance
Specialisation-/Generalisation relationship between classes

Person
name: String
e-mail: String
startWork(t: Time)

Car
0..* repairs mechanic

Person 1 repairs mechanic 0..* Car

Car

SportsCar

Van

Classes

- **Person**
  - **name**: String
  - **e-mail**: String
**Classes**

For class $C$ let $I(C) \neq \emptyset$ be set of objects

"The objects that can have static type $C"
The *Null Type*
The Null Type

Semantics of Null

Each class diagram contains implicitly the Null class

No attributes, no operations

\[ I(\text{Null}) = \{\text{null}\} \]

\[ \text{null} \in I(C) \text{ for any class } C \]

“null is typeable with any type \( C \)”
Subclasses
Subclasses

Semantics of Subclasses

**subclass relation:** \( I(ShortPaper) \subseteq I(Paper) \)

**constraint:** \( I(ShortPaper) \cap I(LongPaper) = \{\text{null}\} \)
Attributes

class name

Person

name: String
e-mail: String

attribute names

attribute types

name compartment

attribute compartment
Attributes

Semantics of Attributes

\( I(name) \) is function from \( I(Person) \) to \( I(String) \)

\( I(name)(aPerson) \) gives a string or null

Always \( f(null) = null \)
Class (Static) Attributes

<table>
<thead>
<tr>
<th>Paper</th>
</tr>
</thead>
<tbody>
<tr>
<td>authors[*]:Person</td>
</tr>
<tr>
<td>number:Integer</td>
</tr>
<tr>
<td>totalnumber:Integer</td>
</tr>
</tbody>
</table>
Semantics

$I(\text{totalnumber})$ is an element of (not a function to) $I(\text{Integer})$

(i.e., Paper.totalnumber is a constant)
## Multiplicities

**Semantics**

<table>
<thead>
<tr>
<th>$M$</th>
<th>$I(M)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0..1</td>
<td>{0, 1}</td>
</tr>
<tr>
<td>0..*</td>
<td>$\mathbb{N}$</td>
</tr>
<tr>
<td>*</td>
<td>$\mathbb{N}$</td>
</tr>
<tr>
<td>1..3</td>
<td>{1, 2, 3}</td>
</tr>
<tr>
<td>7</td>
<td>{7}</td>
</tr>
<tr>
<td>15..19</td>
<td>{15, 16, 17, 18, 19}</td>
</tr>
<tr>
<td>1..3, 7, 15..19</td>
<td>{1, 2, 3, 7, 15, 16, 17, 18, 19}</td>
</tr>
</tbody>
</table>

(i.e., the separator "," acts as set theoretic union)
Associations

Person

name: String
e-mail: String

Paper

authors[ * ] : Person
number: int

role name
association name

multiplicities

3
referee
review

*
Associations

Semantics

$I(\text{review})$ is a relation between $I(\text{Person}) \setminus \{\text{null}\}$ and $I(\text{Paper}) \setminus \{\text{null}\}$

Multiplicity 3 requires: for all $pap \in I(\text{Paper})$, $\text{card}(\{pers \in I(\text{Person}) \mid \text{review}(pers, pap)\}) = 3$
Role Names (Directed Associations)

<table>
<thead>
<tr>
<th>Person</th>
<th>Paper</th>
</tr>
</thead>
<tbody>
<tr>
<td>name: String</td>
<td>authors[*]: Person</td>
</tr>
<tr>
<td>e-mail: String</td>
<td>number: Integer</td>
</tr>
</tbody>
</table>

- 3 \( \leq \) referee \( \rightarrow \) \{ordered\} \( \geq \) review

- multiplicities

- role name
- association name
- property is ordered
Role Names (Directed Associations)

Semantics

\[ I(\text{referee}) : I(\text{Paper}) \rightarrow \text{Set}(I(\text{Person})) \]
\[ I(\text{paper}) : I(\text{Person}) \rightarrow \text{Sequence}(I(\text{Paper})) \]

(supplier multiplicity \( \neq 1 \))

(default role name = client)
Role Names Cont’d

Semantics of role names compatible with association semantics

\[ I(\text{referee})(aPaper) = \{ aPerson \mid \langle aPerson, aPaper \rangle \in I(\text{review}) \} \]
Snapshots

A snapshot of a given class diagram $\mathcal{D}$ is a particular semantics $I$ of $\mathcal{D}$

- **UML object diagram** (for $\mathcal{D}$) including
  - for each class $C$: objects $I(C)$ typeable with $C$
  - maps $I(a): I(C) \rightarrow I(C')$ for all attributes $a$ of type $C'$ in class $C$
  - association instances (pairs) in $I(C) \setminus \{\text{null}\} \times I(C') \setminus \{\text{null}\}$

- an interpretation for operations/methods (Java: independent of snapshot)

- (standard) interpretation of predefined primitive data types and their operations (Integer, String, ...
Snapshots

A **snapshot** of a given class diagram $\mathcal{D}$ is a particular semantics $I$ of $\mathcal{D}$

- UML object diagram (for $\mathcal{D}$) including
  - for each class $C$: objects $I(C)$ typeable with $C$
  - maps $I(a) : I(C) \rightarrow I(C')$ for all attributes $a$ of type $C'$ in class $C$
  - association instances (pairs) in $I(C) \setminus \{\text{null}\} \times I(C') \setminus \{\text{null}\}$

- an interpretation for operations/methods (Java: independent of snapshot)

- (standard) interpretation of predefined primitive data types and their operations (Integer, String, ...)

**Multiplicities and constraints restrict set of admissible snapshots**
Non-admissible (and unintended) instance — Why?
Non-admissible (and unintended) instance — Why?

How to exclude that the author of a paper is its own reviewer?
### Operations: Queries

<table>
<thead>
<tr>
<th>Class Name</th>
<th>Attributes</th>
<th>Operation Name &amp; Signature</th>
</tr>
</thead>
<tbody>
<tr>
<td>Paper</td>
<td>authors[*]:Person, number:Integer</td>
<td>evaluate(), getNumber():Integer{query}</td>
</tr>
</tbody>
</table>
Operations: Queries

Semantics of queries (side-effect free operations, aka pure methods)

Function from owner and parameter classes to result class

\[ I(\text{getNumber}()) : I(\text{Paper}) \rightarrow I(\text{Integer}) \]
Operations: Queries

Semantics of queries (side-effect free operations, aka pure methods)

Function from owner and parameter classes to result class

\[ I(getNumber()) : I(Paper) \rightarrow I(Integer) \]

Semantics of static queries omits owner class argument.
Operations with Side Effects

class name

attributes

operation name & signature

Paper

authors[*]:Person
number:Integer
evaluate()
getNumber():Integer{query}
Operations with Side Effects

Semantics (operations w/o result)

Transition from snapshot to snapshot
(relation between sets of snapshots)
Semantics (operations w/o result)

Transition from snapshot to snapshot (relation between sets of snapshots)

We are not more specific for now: only queries allowed in OCL
Abstract Classes

```
Paper
  authors[*]:Person
  number:Integer

ShortPaper

LongPaper
```
Abstract Classes

Semantics of Abstract Classes (and Interfaces)

\[ I(Paper) = I(ShortPaper) \cup I(LongPaper) \]
\[ I(ShortPaper) \cap I(LongPaper) = \{\text{null}\} \]

No “direct” elements in \( I(Paper) \)
**Semantics**

\(I(\text{Report}) \text{ is a subset of } I(\text{Person}) \setminus \{\text{null}\} \times I(\text{Paper}) \setminus \{\text{null}\}\)
## Data Types

### Integer

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>=(i2:Integer)</code></td>
<td>Boolean</td>
</tr>
<tr>
<td><code>+(i2:Integer)</code></td>
<td>Integer</td>
</tr>
<tr>
<td><code>+(i2:Real)</code></td>
<td>Real</td>
</tr>
<tr>
<td><code>-(i2:Integer)</code></td>
<td>Integer</td>
</tr>
<tr>
<td><code>-(i2:Real)</code></td>
<td>Real</td>
</tr>
<tr>
<td><code>(i2:Integer)</code></td>
<td>Integer</td>
</tr>
<tr>
<td><code>(i2:Real)</code></td>
<td>Real</td>
</tr>
<tr>
<td><code>/ (i2:Integer)</code></td>
<td>Real</td>
</tr>
<tr>
<td><code>/ (i2:Real)</code></td>
<td>Real</td>
</tr>
<tr>
<td><code>abs</code></td>
<td>Integer</td>
</tr>
<tr>
<td><code>div (i2:Integer)</code></td>
<td>Integer</td>
</tr>
<tr>
<td><code>mod (i2:Integer)</code></td>
<td>Integer</td>
</tr>
<tr>
<td><code>max (i2:Integer)</code></td>
<td>Integer</td>
</tr>
<tr>
<td><code>min (i2:Integer)</code></td>
<td>Integer</td>
</tr>
</tbody>
</table>

### String

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>=(i2:String)</code></td>
<td>Boolean</td>
</tr>
<tr>
<td><code>size</code></td>
<td>Integer</td>
</tr>
<tr>
<td><code>concat (string2:String)</code></td>
<td>String</td>
</tr>
<tr>
<td><code>toUpper (string2:String)</code></td>
<td>String</td>
</tr>
<tr>
<td><code>toLower (string2:String)</code></td>
<td>String</td>
</tr>
<tr>
<td><code>substring (lower:Integer, upper:Integer)</code></td>
<td>String</td>
</tr>
</tbody>
</table>
Data Types

Syntax

UML stereotype **data type**

No attributes

Semantics

Semantics \( I(C) \) of a data type class \( C \) fixed for all snapshots \( I \)

\( I(\text{Integer}) \) is the same in all snapshots

All operations are queries (no side effects)

\( I(+) : I(\text{Integer}) \times I(\text{Integer}) \rightarrow I(\text{Integer}) \)
Enumerations

**Person**
- name: String
- e-mail: String

**Paper**
- authors[∗]: Person
- number: Integer

**Report**
- referee
- recommend: Marks

**Marks**
- accept
- reject
- weakly accept
- weakly reject

**Semantics**

Special data type: finite number of instances, explicit representation

\[ I(\text{Marks}) \supseteq \{\text{accept, reject, weakly accept, weakly reject}\} \]
Aggregations

Semantics

Same (formal) semantics as an association

Pragmatics

Part-of relationship, though may be shared with other objects
Compositions

Semantics

Same (formal) semantics as an association

Pragmatics

Owned-by, object lifetime controlled by client (= owner)

Client multiplicity 0..1 or 1
Example: The Composite Pattern

Component

operation()

Composite

operation()
add(c:Component)
remove(c:Component)
getChild(n:Integer)

Leaf

operation()
Use Case Diagrams...

Means for specifying required user scenarios of a system
Use Case Diagrams...

Means for specifying **required** user **scenarios** of a system

Key concepts:

- **Actor**: Role that user plays wrt system; outside the system.

Car Mechanic

22c181: Formal Methods in Software Engineering – p.28/33
Use Case Diagrams...

Means for specifying required user scenarios of a system

Key concepts:

- **Actor**: Role that user plays with respect to the system; outside the system.

- **Use Case**: Set of scenarios subsumed under common user goal
Use Case Diagrams...

Means for specifying **required user scenarios** of a system

Key concepts:

- **Actor**: Role that user plays wrt system; outside the system.

- **Use Case**: Set of scenarios subsumed under common user goal

- **Subject**: System implementing the use case

Connections between use cases, actors and other use cases (stereotypes \«include\» and \«extend\»)
Use Case Syntax

System

User

Privileged User

actor

use case

participation

system boundary

Create File

Open File

Try to open non-existing file

Open by name

Open by hyperlink

Browse file

generalization

≪ precedes ≫

≪ extend ≫

≪ include ≫
Use Case Relations and Stereotypes

Relations

**Participation**: communication of instances of actor/use case
Can be directed (even towards actor) if communication one-way

**Generalization (actors)**: communicates with at most as many use cases

**Generalization (use cases)**: more general case
Use Case Relations and Stereotypes

Relations

**Participation:** communication of instances of actor/use case
Can be directed (even towards actor) if communication one-way

**Generalization (actors):** communicates with at most as many use cases

**Generalization (use cases):** more general case

Stereotypes

**Include:** behaviour always contains that of included use case

**Extend:** behaviour can be augmented with that of extending use case

Much confusion about precise semantics — **discouraged**

**Precedes:** user-defined stereotype, not UML standard
**Use Cases — Important Issues**

**Actors** usually persons, but can be also (other) system

Use relations *among* use cases and *among* actors with great care

**Danger:** clutter up use cases with detailed analysis

**Danger:** Deal with implementation aspects by «extend»

Use cases can be effective help in requirements analysis

Use cases can be bad and confusing “programming language”
Sequence Diagrams

Interaction

Scenario realised by system run

mechanic: Person

aCar: Car

aBulb: Bulb

checkBulbs

repairBulbs

turnOn

22c181: Formal Methods in Software Engineering – p.32/33
Sequence Diagrams

**Interaction**
Scenario realised by system run

**Lifeline**
Individual participant in interaction; instance of class or actor

```
mechanic: Person  aCar: Car  aBulb: Bulb

checkBulbs
repairBulbs

turnOn
```
Sequence Diagrams

**Interaction**
Scenario realised by system run

**Lifeline**
Individual participant in interaction; instance of class or actor

**Message**
Individual communication between lifelines of interaction

```
mechanic: Person
checkBulbs
repairBulbs

aCar: Car
turnOn

aBulb: Bulb
```
Sequence Diagrams

**Interaction**
Scenario realised by system run

**Lifeline**
Individual participant in interaction; instance of class or actor

**Message**
Individual communication between lifelines of interaction

**Execution Occurrence**
Segment of lifeline being “active”

- mechanic: Person
- aCar: Car
- aBulb: Bulb

- checkBulbs
- repairBulbs
- turnOn
Sequence Diagrams (cont’d)

- **Sequence diagrams model communication in ONE scenario**
- **Sequence diagrams refine (part of) ONE use case**
- **Different scenarios require different sequence diagrams**
- **Sequence diagrams are not algorithms!**