Contents & Goals

Last Lecture:
• (Mostly) completed discussion of modelling structure.

This Lecture:
• Educational Objectives:
  Capabilities for following tasks/questions.
• What's the purpose of a behavioural model?
• What does this State Machine mean? What happens if I inject this event?
• Can you please model the following behaviour.

Content:
• For completeness: Modelling Guidelines for Class Diagrams
• Purposes of Behavioural Models
• UML Core State Machines

Design Guidelines for (Class) Diagram (partly following Ambler (2005))
5.1 General Guidelines

- Indicate Visibility Only on Design Models (in contrast to analysis models)

5.2 Class Style Guidelines

- Prefer Complete Singular Nouns for Class Names
- Name Operations with Strong Verbs
- Do Not Model Scaffolding Code [Except for Exceptions]

5.3 Relationships

- Model Relationships Horizontally
- Model a Dependency When the Relationship is Transitory
- Always Indicate the Multiplicity
- Avoid Multiplicity "∗"
- Replace Relationship Lines with Attribute Types

5.4 Associations

- Indicate Role Names When Multiple Associations Between Two Classes Exist
- Make Associations Bidirectional Only When Collaboration Occurs in Both Directions
- Avoid Indicating Non-Navigability
- Question Multiplicities Involving Minimums and Maximums

5.6 Aggregation and Composition

Example: Modelling Games

Task: Game Development

- Genre: Racing
- Rest: open, i.e., Degrees of freedom:
  - Exemplary choice: 2D-Tron
  - Simulation vs. Arcade
  - Platform (SDK or not, open or proprietary, hardware capabilities...)
  - Graphics (3D, 2D, ...)
  - Number of players, AI
    - min. 2, AI open
  - Controller
    - open (later determined by platform)
  - Game experience
    - minimal: main menu and game
Modelling Structure: 2D-Tron

• arcade
• platform open
• 2D
• min. 2, AI open
• controller open
• only game, no menus

In many domains, there are canonical architectures – and adept readers try to see/find/match this!

For games:

- Main External inputs
  - Keyboard
  - Joystick
  - ...
- Game Logic
  - player scores
  - interface inputs/engine
- (Physics) Engine
  - physical objects
  - collision notification
- Output
  - Graphics (from ASCII to bitmap; native or via API)
  - Sound
  - ...

Tron Joystick?

Control
Player
colour
score
direction
speed

Gameplay

Render

OpenGL?

aalib?

AI?

Segment

x0, y0

x1, y1
colour

Engine

area width

area height

1

∗

notify

update

0

∗

Head

world

1

∗

Conventions:

• default µ is 1
• default ξ is +

What Can Be Purposes of Behavioural Models?

Example: Pre-Image

Image (the UML model is supposed to be the blueprint for a software system).

A description of behaviour could serve the following purposes:

- Require Behaviour.
  - “System definitely does this”
  - “This sequence of inserting money and requesting and getting water must be possible.”
  - (Otherwise the software for the vending machine is completely broken.)

- Allow Behaviour.
  - “System does subset of this”
  - “After inserting money and choosing a drink, the drink is dispensed (if in stock).”
  - (If the implementation insists on taking the money first, that’s a fair choice.)

- Forbid Behaviour.
  - “System never does this”
  - “This sequence of getting both, a water and all money back, must not be possible.”
  - (Otherwise the software is broken.)

Note: the latter two are trivially satisfied by doing nothing...

Constructive Behaviour in UML

UML provides two visual formalisms for constructive description of behaviours:

- Activity Diagrams
- State-Machine Diagrams

We (exemplary) focus on State-Machines because

- somehow “practice proven” (in different flavours),
- prevalent in embedded systems community,
- indicated useful by Dobing and Parsons (2006) survey, and
- Activity Diagram’s intuition changed (between UML 1.x and 2.x) from transition-system-like to petri-net-like...

- Example state machines:

  s1
  s2
  s3

  E [ n ≠ ∅ ]
  / x := x + 1;
  n ! F
  / n := 0

  s1
  s2
  F / p ! F

Which Can the Properties of Behavioural Models?
**UML State Machines: Syntax**

(i) **UML State Machine Diagrams.**

(ii) **Definition.** A tuple \( S = (T, C, V, \text{atr}, E) \), \( E \) a set of signals, is called signature (with signals) if and only if \((T, C \cup E, V, \text{atr})\) is a signature (as before).

(iii) **Definition.** Core state machine.

(iv) Map UML State Machine Diagrams to core state machines.

**Semantics:**

(v) **Definition.** Ether (aka. event pool).

(vi) **Definition.** System configuration.

(vii) **Definition.** Event.

(viii) **Definition.** Transformer.

(ix) **Definition.** Transition system, computation.

(x) Transition relation induced by core state machine.

(xi) **Definition.** step, run-to-completion step.

(xii) Later: Hierarchical state machines.

---

**Roadmap:** Chronologically

- **Brief History:**
  - Rooted in Moore/Mealy machines, Transition Systems, etc.
  - Manifest in tool Statemate Harel et al. (1990) (simulation, code-generation);
  - Nowadays also in Matlab/Simulink, etc.
  - From UML 1.x on: State Machines (not the official name, but understood: UML-Statecharts)
  - Late 1990’s: tool Rhapsody with code-generation for state machines.

- **Note:** there is a common core, but each dialect interprets some constructs subtly
  - Crane and Dingel (2007).

---

**Syntax:**

- (i) UML State Machine Diagrams.
- (ii) Def.: Signature with signals.
- (iii) Def.: Core state machine.
- (iv) Map UML State Machine Diagrams to core state machines.

**Semantics:**

- (v) Def.: Ether (aka. event pool).
- (vi) Def.: System configuration.
- (vii) Def.: Event.
- (viii) Def.: Transformer.
- (ix) Def.: Transition system, computation.
- (x) Transition relation induced by core state machine.
- (xi) Def.: step, run-to-completion step.
- (xii) Later: Hierarchical state machines.
In the following, we assume that State-Machines belong to Classes. We will see later that this choice does no harm semantically.

We don't consider multiple state machines: we will see later that this case is similar, but

In the standard

the instances

describes the behaviour of

SM of class diagrams and

Abbreviations and Defaults in the Standard

Expressions

<table>
<thead>
<tr>
<th>Annotation</th>
<th>SM of a UML model consists of a set</th>
</tr>
</thead>
<tbody>
<tr>
<td>( C \in S (\in \mathcal{C}) )</td>
<td>( \text{Act} \in \mathcal{A} ), assumed to be in ( { \text{skip} } ), ( \text{true} ) }: ( \text{Expression}: )</td>
</tr>
</tbody>
</table>

Abbreviations and Defaults in the Standard


