Contents & Goals

Last Lecture:
- Live Sequence Charts Semantics

This Lecture:
- Educational Objectives:
  - Capabilities for following tasks/questions.
  - What's the Liskov Substitution Principle?
  - What is late/early binding?
  - What is the subset, what the uplink semantics of inheritance?
  - What's the effect of inheritance on LSCs, State Machines, System States?
  - What's the idea of Meta-Modelling?

- Content:
  - Quickly complete State Machine semantics
  - Inheritance in UML: concrete syntax
  - Liskov Substitution Principle—desired semantics
  - Two approaches to obtain desired semantics

The Concept of History, and Other Pseudo-States

- History and Deep History: By Example

  - $s_0$
  - $s_1$
  - $s_2$
  - $s_3$
  - $s_b$
  - $s_4$
  - $s_5$

Junction and Choice

- Junction ("static conditional branch"):
  - $[gd_1] / act_1$
  - $[gd_2] / act_2$

- Choice ("dynamic conditional branch"): may get stuck
  - Enters the transition without knowing whether there's an enabled path
  - At best, use "else" and convince yourself that it cannot get stuck
  - Maybe even better: avoid

Entry and Exit Point, Submachine State, Terminate

- Hierarchical states can be "folded" for readability.
  (but: this can also hinder readability.)
- Can even be taken from a different state-machine for re-use.
Entry and Exit Point, Submachine State, Terminate

Hierarchical states can be "folded" for readability. (But: this can also hinder readability.)

Can even be taken from a different state-machine for re-use.

Entry/exit points,

Provide connection points for finer integration into the current level, than just via initial state.

Semantically a bit tricky:

- First the exit action of the exiting state,
- then the actions of the transition,
- then the entry actions of the entered state,
- then the action of the transition from the entry point to an internal state,
- and then that internal state’s entry action.

Terminate Pseudo-State

When a terminate pseudo-state is reached, the object taking the transition is immediately killed.

Deferred Events in State-Machines

Deferred Events: Idea

For ages, UML state machines comprise the feature of deferred events. The idea is as follows:

- Consider the following state machine:

  s₁
  E/F
  s₂
  E/F
  s₃

- Assume we’re stable in s₁, and F is ready in the ether.

- In the framework of the course, F is discarded.

- But we may find it a pity to discard the event and may want to remember it for later processing, e.g. in s₂, in other words, defer it.

General options to satisfy such needs:

- Provide a pattern how to "program" this (uses self-loops and helper attributes).
- Turn it into an original language concept. (← OMG’s choice)

Deferred Events: Syntax and Semantics

Syntactically,

- Each state has (in addition to the name) a set of deferred events.

Default: the empty set.

The semantics is a bit intricate, something like:

- If an event E is dispatched,
- and there is no transition enabled to consume E,
- and E is in the deferred set of the current state configuration,
- then stuff E into some "deferred events space" of the object, (e.g. into the ether (extend ε) or into the local state of the object (extend σ))
- and turn attention to the next event.

Not so obvious:

- Is there a priority between deferred and regular events?
- Is the order of deferred events preserved?
- ...

[Fecher and Schönborn, 2007], e.g., claim to provide semantics for the complete Hierarchical State Machine language, including deferred events.

Active and Passive Objects

[Harel and Gery, 1997]
A class is either passive or reactive. Which combinations do we understand? [Harel and Gery, 1997]

And when should we decide that a class is reactive implies active?

Delegate all dispatching of events to the active objects. In UML, the former is called signal.

And that there are events in the ether for all three.

Reasonable Approaches:

- Firstly, establish that each object has (in the operating systems sense) an own thread:
  - Its Act
- ... and waits until ...
  - asks
- ... to ...
  - e

Events for active objects.

Passive and Reactive Classes

Active Reactive Classes

What About Methods?

Create an instance and waits until ...

Events for active objects.

And when should we decide that a class is reactive implies active?

Reasonable Approaches:

- Firstly, establish that each object has (in the operating systems sense) an own thread:
  - Its Act
- ... and waits until ...
  - asks
- ... to ...
  - e

Events for active objects.

Passive and Reactive Classes

Active Reactive Classes

What About Methods?

Create an instance and waits until ...

Events for active objects.

And when should we decide that a class is reactive implies active?

Reasonable Approaches:

- Firstly, establish that each object has (in the operating systems sense) an own thread:
  - Its Act
- ... and waits until ...
  - asks
- ... to ...
  - e

Events for active objects.

Passive and Reactive Classes

Active Reactive Classes

What About Methods?

Create an instance and waits until ...

Events for active objects.

And when should we decide that a class is reactive implies active?

Reasonable Approaches:

- Firstly, establish that each object has (in the operating systems sense) an own thread:
  - Its Act
- ... and waits until ...
  - asks
- ... to ...
  - e

Events for active objects.

Passive and Reactive Classes

Active Reactive Classes

What About Methods?

Create an instance and waits until ...

Events for active objects.

And when should we decide that a class is reactive implies active?

Reasonable Approaches:

- Firstly, establish that each object has (in the operating systems sense) an own thread:
  - Its Act
- ... and waits until ...
  - asks
- ... to ...
  - e

Events for active objects.

Passive and Reactive Classes

Active Reactive Classes

What About Methods?

Create an instance and waits until ...

Events for active objects.

And when should we decide that a class is reactive implies active?

Reasonable Approaches:

- Firstly, establish that each object has (in the operating systems sense) an own thread:
  - Its Act
- ... and waits until ...
  - asks
- ... to ...
  - e

Events for active objects.

Passive and Reactive Classes

Active Reactive Classes

What About Methods?

Create an instance and waits until ...

Events for active objects.

And when should we decide that a class is reactive implies active?

Reasonable Approaches:

- Firstly, establish that each object has (in the operating systems sense) an own thread:
  - Its Act
- ... and waits until ...
  - asks
- ... to ...
  - e

Events for active objects.

Passive and Reactive Classes

Active Reactive Classes

What About Methods?
Behavioural Features: Visibility and Properties

Visibility:
- Extend typing rules to sequences of actions such that a well-typed action sequence only calls visible methods.

Useful properties:
- Concurrency (isthread-safe)
- Guarded (some mechanism ensures / should ensure mutual exclusion)
- Sequential (is not threadsafe, users have to ensure mutual exclusion)
- Is query (doesn’t modify the statespace (thus thread-safe))

For simplicity, we leave the notion of steps untouched, we construct our semantics around state machines. Yet we could explain pre/post in OCL (if we wanted to).

Pessimistic view: They are legion...
- For instance, allow absence of initial pseudo-states can then "be" in enclosing state without being in any substate; or assume one of the children states nondeterministically (implicitly) enforced determinism, e.g. by considering the order in which things have been added to the CASE tool's repository, or graphical order.
- Allow true concurrency.

Exercise: Search the standard for "semantical variation point".

- [Crane and Dingel, 2007], e.g., provide an in-depth comparison of Statemate, UML, and Rhapsody state machines—the bottom line is:
  - the intersection is not empty (i.e. there are pictures that mean the same thing to all three communities)
  - none is the subset of another (i.e. for each pair of communities exist pictures meaning different things).

Optimistic view: tools exist with complete and consistent code generation.
Inheritance: Generalisation Relation

• Alternatives renderings:

C ⊃ D₁ ⊃ D₂

• Read:

C generalises D₁ and D₂;
C isa generalisation of D₁ and D₂,
D₁ and D₂ specialise C;
D₁ isa (specialisation of) C,
D₂ isa C.

• Well-formedness rule: No cycles in the generalisation relation.

Abstract Syntax

Recall: a signature (with signals) is a tuple

/CB = (CC, BV, V, atr, BX)

Now (finally): extend to

/CB = (CC, BV, V, atr, BX, F, mth, ⊳)

where

F/mth are methods, analogously to attributes and
⊳ ⊆ (BV × BV) ∪ (BX × BX)

isa generalisation relation such that
C ⊃ ⊳ C for no C ∈ BV ("acyclic").

C ⊃ D reads as

• C is a generalisation of D,
• D is a specialisation of C,
• D inherits from C,
• D is a sub-class of C,
• C is a super-class of D,
...

Mapping Concreteto Abstract Syntax by Example

C₀ x : Int
C₁ x : Int
C₂

Note: we can have multiple inheritance.

Reflexive, Transitive Closure of Generalisation

Definition.

Given classes C₀, C₁, D ∈ BV, we say D inherits from C₀ via C₁ if and only if there are C₁₀, . . .Cₙ₀, C₁₁, . . .Cₘ₁ ∈ BV such that
C₀ ⊃ C₁₀ ⊃ . . . ⊃ Cₙ₀ ⊃ C₁ ⊃ . . . ⊃ Cₘ₁ ⊃ D.

We use '⪯' to denote the reflexive, transitive closure of '⊳'.

In the following, we assume

• that all attribute (method) names are of the form
C :: v, C ∈ BV ∪ BX (C :: f, C ∈ BV)
• that we have C :: v ∈ atr (C) resp. C :: f ∈ mth (C) if and only if v (f) appears in an attribute (method) compartment of C in a class diagram.

We still want to accept "context C in v:
< v₀", which v is meant? Later!

Desired Semanticsof Specialisation: Subtyping

There is a classical description of what one expects from sub-types, which in the OO domain is closely related to inheritance:

The principle of typesubstitutability [Liskov, 1988, Liskov and Wing, 1994].

(Liskov Substitution Principle (LSP).)

"If foreach object o₁ of type S there is an object o₂ of type T such that for all programs P defined in terms of T, the behavior of P is unchanged when o₁ is substituted for o₂ then S is a subtype of T."

S sub-type of T:
⇐⇒ ∀ o₁ ∈ S ∃ o₂ ∈ T ∀ P T • P T (o₁) = P T (o₂)
References


