Software Design, Modelling and Analysis in UML
Lecture 15: Hierarchical State Machines I

Contents & Goals

Last Lecture:
• RTC-Rules: Discard, Dispatch, Commence. item Step, RTC

This Lecture:
• Educational Objectives:
  • What does this State Machine mean? What happens if I inject this event?
  • Can you please model the following behaviour.
  • What is: initial state.
  • What is: AND-State, OR-State, pseudo-state, entry/exit/do, final state, ...

• Content:
  • Transformer: Create and Destroy, Divergence
  • Putting It All Together
  • Hierarchical State Machines Syntax

Transformer: Create

abstract syntax concrete syntax
create (C, expr, v)

intuitive semantics
Create an object of class C and assign it to attribute v of the object denoted by expression expr.

well-typedness
expr : τ D, v ∈ atr(D), atr(C) = {⟨v1 : τ1, expr0i⟩ | 1 ≤ i ≤ n}

semantics
observables
(error) conditions
I llbracket expr0i rrbracket (σ, ux) not defined for some i.

• We use an “and assign”-action for simplicity — it doesn’t add or remove expressive power, but moving creation to the expression language raises all kinds of other problems such as order of evaluation (and thus creation).
• Also for simplicity: no parameters to constructor (∼ parameters of constructor).

Adding them is straightforward (but somewhat tedious).

Create Transformer Example

SM C:

s1 . . . s2
n := new C;

create (C, expr, v)

```ini
create (C, expr, v)[ux](σ, ε) = ...
```

σ : d : D

```ini
n = ∅:
```

ε : ε'
Abstract syntax concrete syntax

Create Transformer Example


destroy...

x ignored is destroyed...
Example

Transformer: Destroy
What people may dislike on our definition of RTC-step is that it takes a global and non-compositional view. That is:

- In the projection onto a single object we still see the effect of interaction with other objects.
- Adding classes (or even objects) may change the divergence behaviour of existing ones.
- Compositional would be: the behaviour of a set of objects is determined by the behaviour of each object “in isolation”. Our semantics and notion of RTC-step doesn’t have this (often desired) property.

Can we give (syntactical) criteria such that any global run-to-completion step is an interleaving of local ones?

Maybe:

- **(A)** Refer to private features only via “self”. (Recall that other objects of the same class can modify private attributes.)
- **(B)** Let objects only communicate by events, i.e. don’t let them modify each other’s local state via links at all.

References