Software Design, Modelling and Analysis in UML

Lecture 11: Core State Machines I

2014-12-04

Prof. Dr. Andreas Podelski, **Dr. Bernd Westphal**

Albert-Ludwigs-Universität Freiburg, Germany

Contents & Goals

Last Lecture:

- Associations (up to some rest)

This Lecture:

- **Educational Objectives:** Capabilities for following tasks/questions.
  - What does this State Machine mean? What happens if I inject this event?
  - Can you please model the following behaviour.
  - What is: Signal, Event, Ether, Transformer, Step, RTC.
- **Content:**
  - Associations cont'd, back to main track
  - Core State Machines
  - UML State Machine syntax

Associations: The Rest

The Rest

Recapitulation: Consider the following association:

$$\langle r : \langle role_1 : C_1, \mu_1, P_1, \xi_1, \nu_1, o_1 \rangle, \ldots, \langle role_n : C_n, \mu_n, P_n, \xi_n, \nu_n, o_n \rangle \rangle$$

- **Association name** r and **role names**/**types** $role_i/C_i$ induce extended system states λ.
- **Multiplicity** μ is considered in OCL syntax.
- **Visibility** ξ/**Navigability** ν: well-typedness.

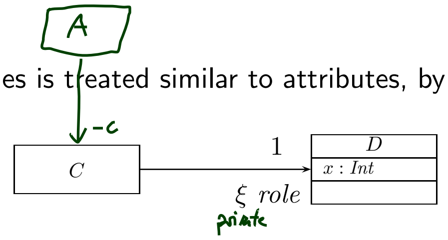
Now the rest:

- **Multiplicity** μ: we propose to view them as constraints.
- **Properties** P_i: even more typing.
- **Ownership** o: getting closer to pointers/references.
- **Diamonds**: exercise.

Visibility

Visibility of role-names is treated similar to attributes, by **typing rules**.

Question: given



self$_A$.c OK

self$_A$.c.role NOT OK

self$_C$.c OK

is the following OCL expression well-typed or not (wrt. visibility):

$$\text{context } C \text{ inv} : \mathit{self}.\mathit{role}.x > 0$$

Basically the same rule as before (similar for other multiplicities):

$\mathit{role}(w) \quad : \tau_C \to \tau_D \qquad \mu = 0..1$ or $\mu = 1$,

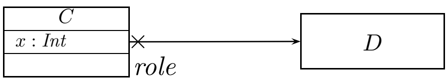
$\mathit{role}(\mathit{expr}_1(w)) : \tau_C \to \tau_D \qquad \mu = 0..1$ or $\mu = 1$, $\mathit{expr}_1(w) : \tau_C$, $w : \tau_{C_1}$, and $C_1 = C$ or $\xi = +$

$$\langle r : \ldots \langle \mathit{role} : D, \mu, \_, \xi, \_, \_\rangle, \ldots \langle \mathit{role}' : C, \_, \_, \_, \_, \_\rangle, \ldots \rangle \in V$$

Navigability

Navigability is similar to visibility: expressions over non-navigable association ends ($\nu = \times$) are **basically** type-correct, but **forbidden**.

Question: given



is the following OCL expression well-typed or not (wrt. navigability)?

$$\text{context } D \text{ inv} : \mathit{self}.\mathit{role}.x > 0$$

The standard says: navigation is...

- '$-$': ...possible
- '$>$': ...efficient
- '$\times$': ...not possible

So: In general, UML associations are different from pointers/references!

But: Pointers/references can faithfully be modelled by UML associations.

The Rest of the Rest

Recapitulation: Consider the following association:

$$\langle r : \langle role_1 : C_1, \mu_1, P_1, \xi_1, \nu_1, o_1 \rangle, \ldots, \langle role_n : C_n, \mu_n, P_n, \xi_n, \nu_n, o_n \rangle \rangle$$

- **Association name** r and **role names**/**types** $role_i / C_i$ induce extended system states λ.
- **Multiplicity** μ is considered in OCL syntax.
- **Visibility** ξ/**Navigability** ν: well-typedness.

Now the rest:

- **Multiplicity** μ: we propose to view them as constraints.
- **Properties** P_i: even more typing.
- **Ownership** o: getting closer to pointers/references.
- **Diamonds**: exercise.

Multiplicities as Constraints

Recall: The multiplicity of an association end is a term of the form:

$$\mu ::= * \mid N \mid N..M \mid N..* \mid \mu, \mu \qquad (N, M \in \mathbb{N})$$

Proposal: View multiplicities (except $0..1$, 1) as additional invariants/constraints.

Recall: we can normalize each multiplicity μ to the form

$$\mu = \underbrace{N_1..N_2}, \ \ldots, \ \underbrace{N_{2k-1}..N_{2k}}$$

where $N_i \leq N_{i+1}$ for $1 \leq i \leq 2k$, $\quad N_1, \ldots, N_{2k-1} \in \mathbb{N}$, $\quad N_{2k} \in \mathbb{N} \cup \{*\}$.

Multiplicities as Constraints

$$\mu = N_1..N_2, \;\ldots,\; N_{2k-1}..N_{2k}$$

where $N_i \leq N_{i+1}$ for $1 \leq i \leq 2k$, $\quad N_1, \ldots, N_{2k-1} \in \mathbb{N}$, $\quad N_{2k} \in \mathbb{N} \cup \{*\}$.

Define $\mu^C_{\mathsf{OCL}}(role) :=$ context C inv :

$$(N_1 \leq role \texttt{->} \mathsf{size}() \leq N_2) \text{ or } \ldots \text{ or } (N_{2k-1} \leq role \texttt{->} \mathsf{size}() \underbrace{\leq N_{2k}}_{\text{omit if } N_{2k} = *})$$

for each $\mu \neq 0..1$, $\mu \neq 1$,

$$\langle r : \ldots, \langle role : D, \mu, \_, \_, \_, \_\rangle, \ldots, \langle role' : C, \_, \_, \_, \_, \_\rangle, \ldots\rangle \in V \text{ or}$$
$$\langle r : \ldots, \langle role' : C, \_, \_, \_, \_, \_\rangle, \ldots, \langle role : D, \mu, \_, \_, \_, \_\rangle, \ldots\rangle \in V, role \neq role'.$$

For $\mu = 0$: context C inv: oclIsUndefined(role)

And **define**

$$\mu^C_{\mathsf{OCL}}(role) := \text{context } C \text{ inv : not(oclIsUndefined}(role))$$

for each $\mu = 1$.

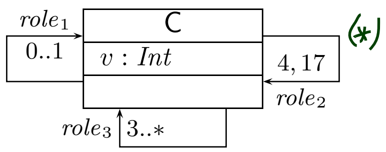
Note: in n-ary associations with $n > 2$, there is redundancy.

Multiplicities as Constraints Example

$\mu^C_{\mathsf{OCL}}(role) =$ context C inv :
$\quad (N_1 \leq role \texttt{->} \mathsf{size}() \leq N_2)$ or $\ldots$ or $(N_{2k-1} \leq role \texttt{->} \mathsf{size}() \leq N_{2k})$

note:
0..1
is equivalent to
0..0, 1..1

$\mathcal{CD}$:



$Inv(\mathcal{CD}) =$

- { context C inv: $4 \leq role_2 \texttt{->} \mathsf{size}() \leq 4$ or $17 \leq role_2 \texttt{->} \mathsf{size}() \leq 17$, (*)
  ... }

  $\{$context C inv : $4 \leq role_2 \texttt{->} \mathsf{size}() \leq 4$ or $17 \leq role_2 \texttt{->} \mathsf{size}() \leq 17\}$
  $= \{$context C inv : $role_2 \texttt{->} \mathsf{size}() = 4$ or $role_2 \texttt{->} \mathsf{size}() = 17\}$
- $\cup \{$context C inv : $3 \leq role_3 \texttt{->} \mathsf{size}()\}$

Why Multiplicities as Constraints?

More precise, can't we just use **types**? (cf. Slide 26)

- $\mu = 0..1$, $\mu = 1$:
  many programming language have direct correspondences (the first corresponds to type pointer, the second to type reference) — therefore treated specially.
- $\mu = *$:
  could be represented by a set data-structure type without fixed bounds — no problem with our approach, we have $\mu_{\mathsf{OCL}} = \mathit{true}$ anyway.
- $\mu = 0..3$:
  use array of size ~~5~~ 3 — if model behaviour (or the implementation) adds 5th identity, we'll get a runtime error, and thereby see that the constraint is violated.
  Principally acceptable, but: checks for array bounds everywhere...?
- $\mu = 5..7$:
  could be represented by an array of size 7 — but: few programming languages/data structure libraries allow lower bounds for arrays (other than 0).
  If we have 5 identities and the model behaviour removes one, this should be a violation of the constraints imposed by the **model**.
  The implementation which does this removal is **wrong**. How do we see this...?

Multiplicities Never as Types...?

Well, if the **target platform** is known and fixed,
and the target platform has, for instance,

- reference types,
- range-checked arrays with positions $0, \dots, N$,
- set types,

then we could simply **restrict** the syntax of multiplicities to

$$\mu ::= 1 \mid 0..N \mid *$$

and don't think about constraints
(but use the obvious 1-to-1 mapping to types)...

In general, **unfortunately**, we don't know.

Multiplicities as Constraints of Class Diagram

Recall/Later:

$\mathscr{CD} = \{CD_1, \dots, CD_n\}$

$[\![\cdot]\!]$

signature $\mathscr{S}(\mathscr{CD})$ — invariants $Inv(\mathscr{CD})$

distinguish

basic (classes and attributes) — extended (visibility)

in a minute

From now on: $Inv(\mathscr{CD}) = \{\text{constraints occurring in notes}\} \cup \{\mu^C_{\mathsf{OCL}}(role) \mid$

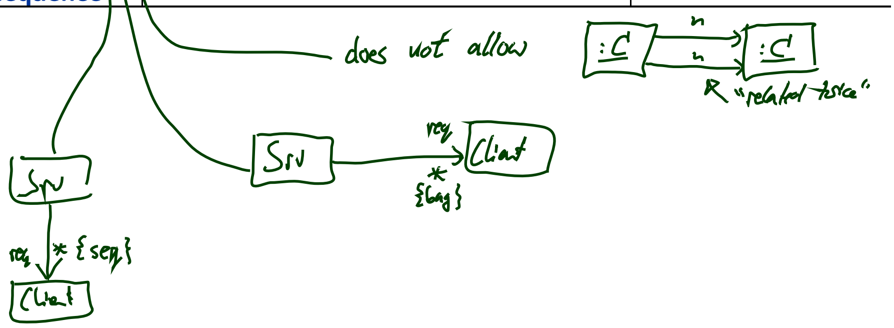
$$\langle r : \dots, \langle role : D, \mu, \_, \_, \_, \_\rangle, \dots, \langle role' : C, \_, \_, \_, \_, \_\rangle, \dots \rangle \in V \text{ or}$$
$$\langle r : \dots, \langle role' : C, \_, \_, \_, \_, \_\rangle, \dots, \langle role : D, \mu, \_, \_, \_, \_\rangle, \dots \rangle \in V,$$
$$role \neq role', \mu \notin \{0..1\}\}.$$

– 11 – 2014-12-04 – Sassocrest –

Properties

We don't want to cover association **properties** in detail,
only some observations (assume binary associations):

Property	Intuition	Semantical Effect
unique	one object has **at most one** r-link to a single other object	**current setting**
bag	one object may have **multiple** r-links to a single other object	have $\lambda(r)$ yield multi-sets
ordered, **sequence**	an r-link is a **sequence** of object identities (possibly including duplicates)	have $\lambda(r)$ yield sequences



– 11 – 2014-12-04 – Sassocrest –

Properties

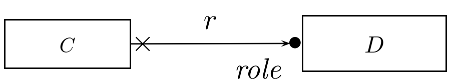
We don't want to cover association **properties** in detail,
only some observations (assume binary associations):

Property	Intuition	Semantical Effect
unique	one object has **at most one** r-link to a single other object	**current setting**
bag	one object may have **multiple** r-links to a single other object	have $\lambda(r)$ yield multi-sets
ordered, **sequence**	an r-link is a **sequence** of object identities (possibly including duplicates)	have $\lambda(r)$ yield sequences

Property	OCL Typing of expression $role(expr)$
unique	$\tau_D \to Set(\tau_C)$
bag	$\tau_D \to Bag(\tau_C)$
ordered, **sequence**	$\tau_D \to Seq(\tau_C)$

For **subsets**, **redefines**, **union**, etc. see [OMG, 2007a, 127].

Ownership



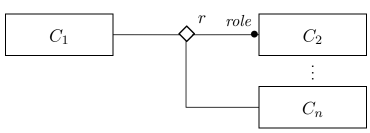
Intuitively it says:

Association r is **not a "thing on its own"** (i.e. provided by λ),
but association end '$role$' is **owned** by C (!).
(That is, it's stored inside C object and provided by σ).

So: if multiplicity of $role$ is $0..1$ or 1, then the picture above is very close to concepts of pointers/references.

Actually, ownership is seldom seen in UML diagrams. Again: if target platform is clear, one may well live without (cf. [OMG, 2007b, 42] for more details).

Not clear to me:



Back to the Main Track

Back to the main track:

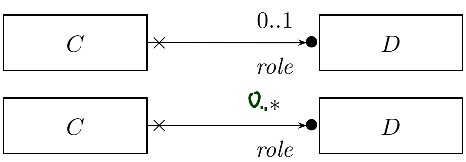
Recall: on some earlier slides we said, the extension of the signature is **only** to study associations in "full beauty"..
For the remainder of the course, we should look for something simpler...

Proposal:

- **from now on**, we only use associations of the form

(i)

(ii)



  (And we may omit the non-navigability and ownership symbols.)

- Form (i) introduces $role : C_{0,1}$, and form (ii) introduces $role : C_*$ in V.
- In both cases, $role \in atr(C)$.
- We drop λ and go back to our nice σ with $\sigma(u)(role) \subseteq \mathscr{D}(D)$.

OCL Constraints in (Class) Diagrams

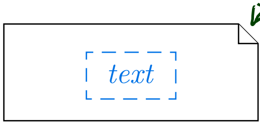
Where Shall We Put OCL Constraints?

Two options:
(0) additional documents
- (i) Notes.
- (ii) Particular dedicated places.

(i) **Notes**:

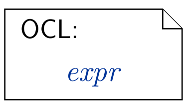
A UML **note** is a picture of the form

Eselsohr
(dog's ear)

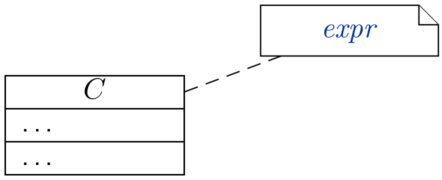


text can principally be **everything**, in particular **comments** and **constraints**.

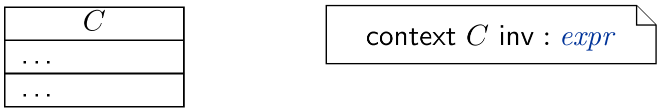
Sometimes, content is **explicitly classified** for clarity:



OCL in Notes: Conventions

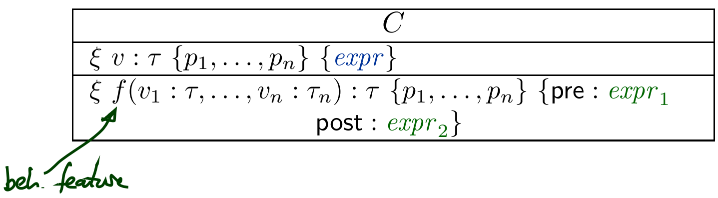


stands for

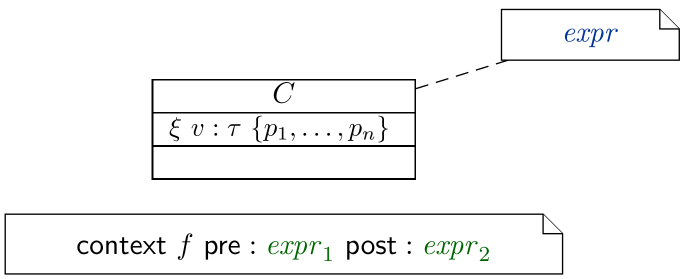


Where Shall We Put OCL Constraints?

(ii) **Particular dedicated places** in class diagrams: (behav. feature: later)



For simplicity, we view the above as an abbreviation for



Invariants of a Class Diagram

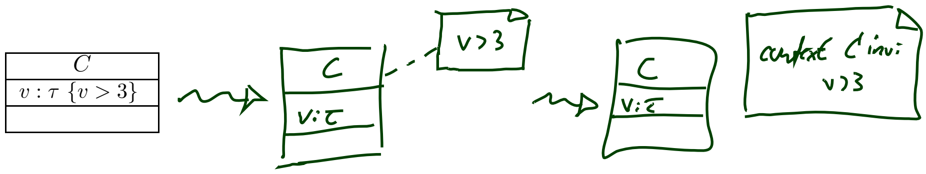
- Let $\mathcal{CD}$ be a class diagram.
- As we (now) are able to recognise OCL constraints when we see them, we can define

$$\mathit{Inv}(\mathcal{CD})$$

  as the set $\{\varphi_1, \ldots, \varphi_n\}$ of OCL constraints **occurring** in notes in $\mathcal{CD}$ — after **unfolding** all abbreviations (cf. next slides).

- As usual: $\mathit{Inv}(\mathscr{CD}) := \bigcup_{\mathcal{CD} \in \mathscr{CD}} \mathit{Inv}(\mathcal{CD})$.
- **Principally clear**: $\mathit{Inv}(\,\cdot\,)$ for any kind of diagram.

Invariant in Class Diagram Example



If $\mathscr{CD}$ consists of only $\mathcal{CD}$ with the single class C, then

- $\mathit{Inv}(\mathscr{CD}) = \mathit{Inv}(\mathcal{CD}) = $ „

Semantics of a Class Diagram

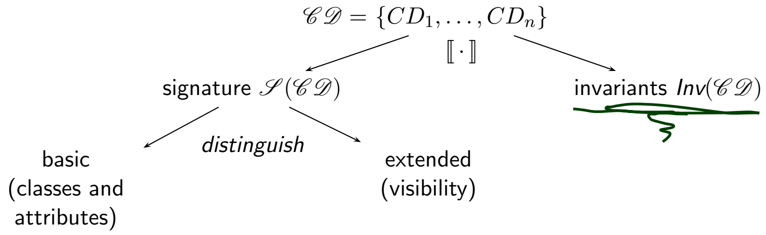
Definition. Let $\mathscr{CD}$ be a set of class diagrams.

We say, the semantics of $\mathscr{CD}$ is the signature it induces and the set of OCL constraints occurring in $\mathscr{CD}$, denoted

$$[\![\mathscr{CD}]\!] := \langle \mathscr{S}(\mathscr{CD}), Inv(\mathscr{CD})\rangle.$$

Given a structure $\mathscr{D}$ of $\mathscr{S}$ (and thus of $\mathscr{CD}$), the class diagrams describe the system states $\Sigma_{\mathscr{S}}^{\mathscr{D}}$, of which **some** may satisfy $Inv(\mathscr{CD})$.

In pictures:



Pragmatics

Recall: a UML **model** is an image or pre-image of a software system.

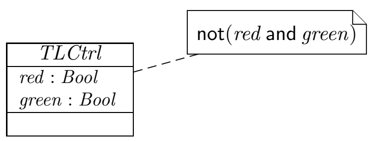
A set of class diagrams $\mathscr{CD}$ with invariants $Inv(\mathscr{CD})$ describes the **structure** of system states.

Together with the invariants it can be used to state:

- **Pre-image**: Dear programmer, please provide an implementation which uses only system states that satisfy $Inv(\mathscr{CD})$.
- **Post-image**: Dear user/maintainer, in the existing system, only system states which satisfy $Inv(\mathscr{CD})$ are used.

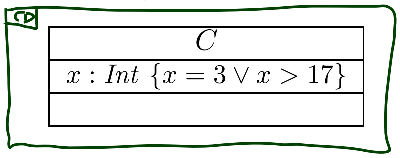
(The exact meaning of "use" will become clear when we study behaviour — intuitively: the system states that are reachable from the initial system state(s) by calling methods or firing transitions in state-machines.)

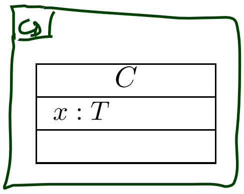
Example: highly abstract model of traffic lights controller.



Constraints vs. Types

Find the 10 differences:





$\mathscr{D}(T) = \{3\}$
$\cup\{n \in \mathbb{N} \mid n > 17\}$

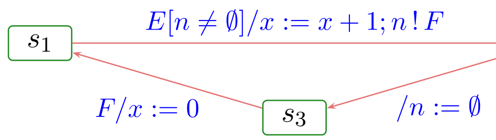
- $x = 4$ is well-typed in the left context, a system state satisfying $x = 4$ violates the constraints of the diagram.
- $x = 4$ is not even well-typed in the right context, there cannot be a system state with $\sigma(u)(x) = 4$ because $\sigma(u)(x)$ is supposed to be in $\mathscr{D}(T)$ (by definition of system state).

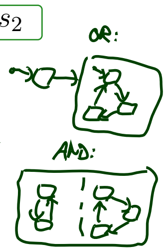
Rule-of-thumb:

- If something **"feels like" a type** (one criterion: has a natural correspondence in the application domain), then make it a type.
- If something is a **requirement** or restriction of an otherwise useful type, then make it a constraint.

UML State Machines

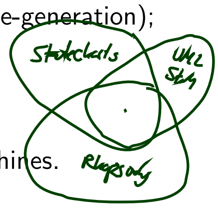
UML State Machines





Brief History:

- Rooted in **Moore/Mealy machines**, Transition Systems
- [Harel, 1987]: **Statecharts** as a concise notation, introduces in particular hierarchical states.
- Manifest in tool **Statemate** [Harel et al., 1990] (simulation, code-generation); nowadays also in **Matlab/Simulink**, etc.
- From UML 1.x on: **State Machines** (in State Chart Diagrams) (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 different [Crane and Dingel, 2007]. *(Would be too easy otherwise. . .)*

Roadmap: Chronologically

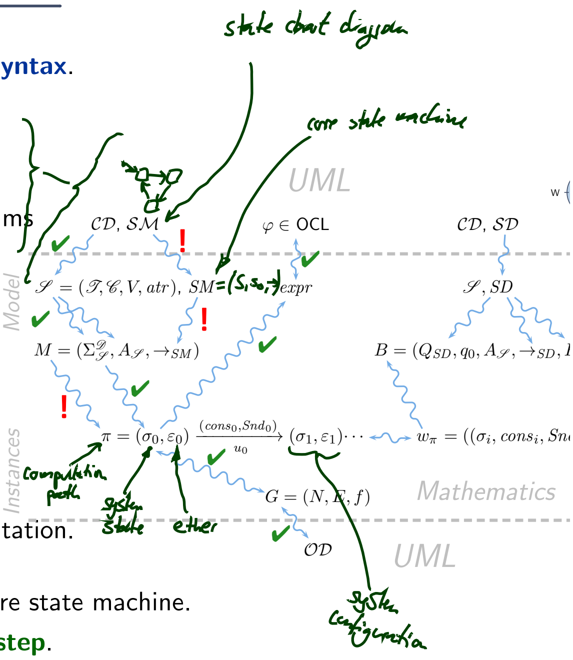
(i) What do we (have to) cover?
UML State Machine Diagrams **Syntax**.

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

Semantics:
The Basic Causality Model

(v) Def.: **Ether** (aka. event pool)
(vi) Def.: **System configuration**.
(vii) Def.: **Event**.
(viii) Def.: **Transformer**.
(ix) Def.: **Transition system**, computation.

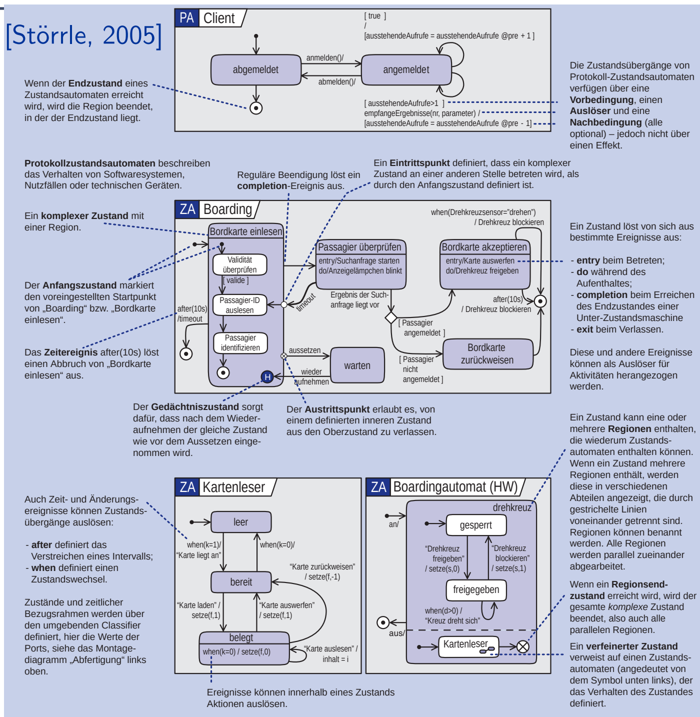
(x) Transition relation induced by core state machine.
(xi) Def.: **step**, **run-to-completion step**.
(xii) Later: Hierarchical state machines.



UML State Machines: Syntax

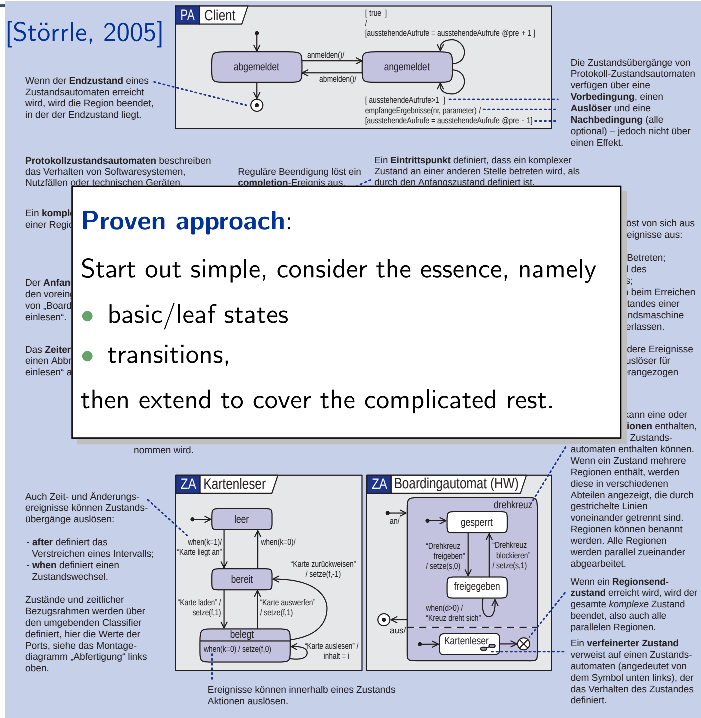
UML State-Machines: What do we have to cover?

[Störrle, 2005]



UML State-Machines: What do we have to cover?

[Störrle, 2005]



Proven approach:

Start out simple, consider the essence, namely

- basic/leaf states
- transitions,

then extend to cover the complicated rest.

Signature With Signals

Definition. A tuple

$$\mathscr{S} = (\mathscr{T}, \mathscr{C}, V, atr, \mathscr{E}), \qquad \mathscr{E} \subseteq \mathscr{C} \text{ a set of signals,}$$

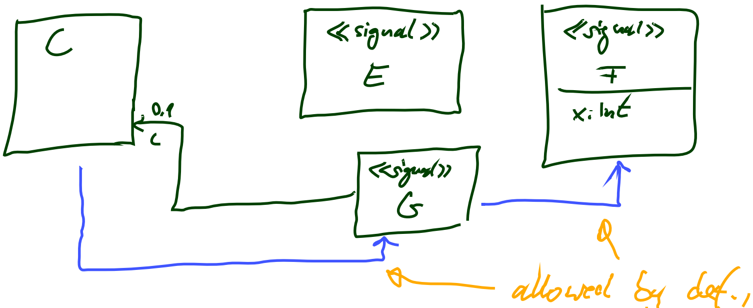
is called signature (with signals) if and only if

$$(\mathscr{T}, \mathscr{C}, V, atr)$$

is a signature (as before).

Note: Thus conceptually, **a signal is a class** and can have attributes of plain type and associations.

Signature With Signals: Example



allowed by def., ruled out later

$$\mathscr{S} = (\mathscr{T}, \{C, E, F, G\}, \{x : Int,\ c : C_{0,1}\}, \{C \mapsto \emptyset, E \mapsto \emptyset, F \mapsto \{x : Int\}, G \mapsto \{c : C_{0,1}\}\}, \{E, F, G\})$$

Core State Machine

Definition.
A core state machine over signature $\mathscr{S} = (\mathscr{T}, \mathscr{C}, V, atr, \mathscr{E})$ is a tuple

$$M = (S, s_0, \rightarrow)$$

where

- S is a non-empty, finite set of **(basic) states**,
- $s_0 \in S$ is an **initial state**,
- and

set of signals source state destination state

$\dot\cup$ "disjoint union" $\_ \notin \mathscr{E}$

$$\rightarrow\ \subseteq S \times \underbrace{(\mathscr{E} \,\dot\cup\, \{\_\})}_{\text{trigger}} \times \underbrace{Expr_{\mathscr{S}}}_{\text{guard}} \times \underbrace{Act_{\mathscr{S}}}_{\text{action}} \times S$$

is a labelled transition relation.

We assume a set $Expr_{\mathscr{S}}$ of boolean expressions (may be OCL, may be something else) and a set $Act_{\mathscr{S}}$ of **actions** over $\mathscr{S}$.

From UML to Core State Machines: By Example

UML state machine diagram $\mathcal{SM}$:

$$annot ::= \Big[\ \ \langle event\rangle [\ `.\text{'}\ \langle event\rangle]^{*}\ \ \ [\ `[\text{'}\ \langle guard\rangle\ `]\text{'}\ ]\ \ \ [\ `/\text{'}\ \langle action\rangle]\ \ \Big]$$

with

- $event \in \mathscr{E}$,
- $guard \in Expr_{\mathscr{S}}$ (default: *true*, assumed to be in $Expr_{\mathscr{S}}$)
- $action \in Act_{\mathscr{S}}$ (default: `skip`, assumed to be in $Act_{\mathscr{S}}$)

maps to

$$M(\mathcal{SM}) = \big(\underbrace{\{s_1, s_2\}}_{S}, \underbrace{s_1}_{s_0}, \underbrace{(s_1, event, guard, action, s_2)}_{\to}\big)$$

References

[Crane and Dingel, 2007] Crane, M. L. and Dingel, J. (2007). UML vs. classical vs. rhapsody statecharts: not all models are created equal. *Software and Systems Modeling*, 6(4):415–435.

[Harel, 1987] Harel, D. (1987). Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8(3):231–274.

[Harel et al., 1990] Harel, D., Lachover, H., et al. (1990). Statemate: A working environment for the development of complex reactive systems. *IEEE Transactions on Software Engineering*, 16(4):403–414.

[OMG, 2007a] OMG (2007a). Unified modeling language: Infrastructure, version 2.1.2. Technical Report formal/07-11-04.

[OMG, 2007b] OMG (2007b). Unified modeling language: Superstructure, version 2.1.2. Technical Report formal/07-11-02.

[Störrle, 2005] Störrle, H. (2005). *UML 2 für Studenten*. Pearson Studium.