

Software Design, Modelling and Analysis in UML

Lecture 12: Core State Machines II

2013-12-09

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

Albert-Ludwigs-Universität Freiburg, Germany

Contents & Goals

Last Lecture:

- State machine syntax
- core state machines

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:**
 - The basic causality model
 - Ether
 - System Configuration, Transformer
 - Examples for transformer
 - Run-to-completion Step

The Basic Causality Model

6.2.3 The Basic Causality Model [OMG, 2007b, 12]

“**Causality model**’ is a specification of how things happen at run time [...].

The causality model is quite straightforward:

- Objects respond to **messages** that are generated by objects executing communication actions.
- When these messages arrive, the receiving objects eventually respond by executing the behavior that is **matched** to that message.
- The dispatching method by which a particular behavior is associated with a given message depends on the higher-level formalism used and is not defined in the UML specification **(i.e., it is a semantic variation point)**.

The causality model also subsumes behaviors invoking each other and passing information to each other through arguments to parameters of the invoked behavior, [...].

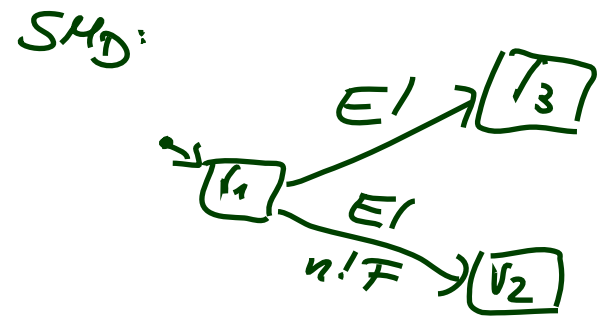
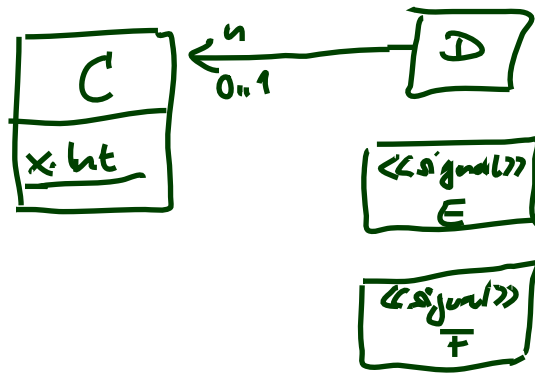
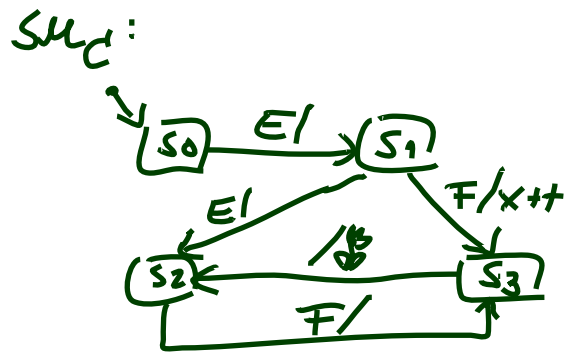
This purely ‘procedural’ or ‘process’ model can be used by itself or in conjunction with the object-oriented model of the previous example.”

15.3.12 StateMachine [OMG, 2007b, 563]

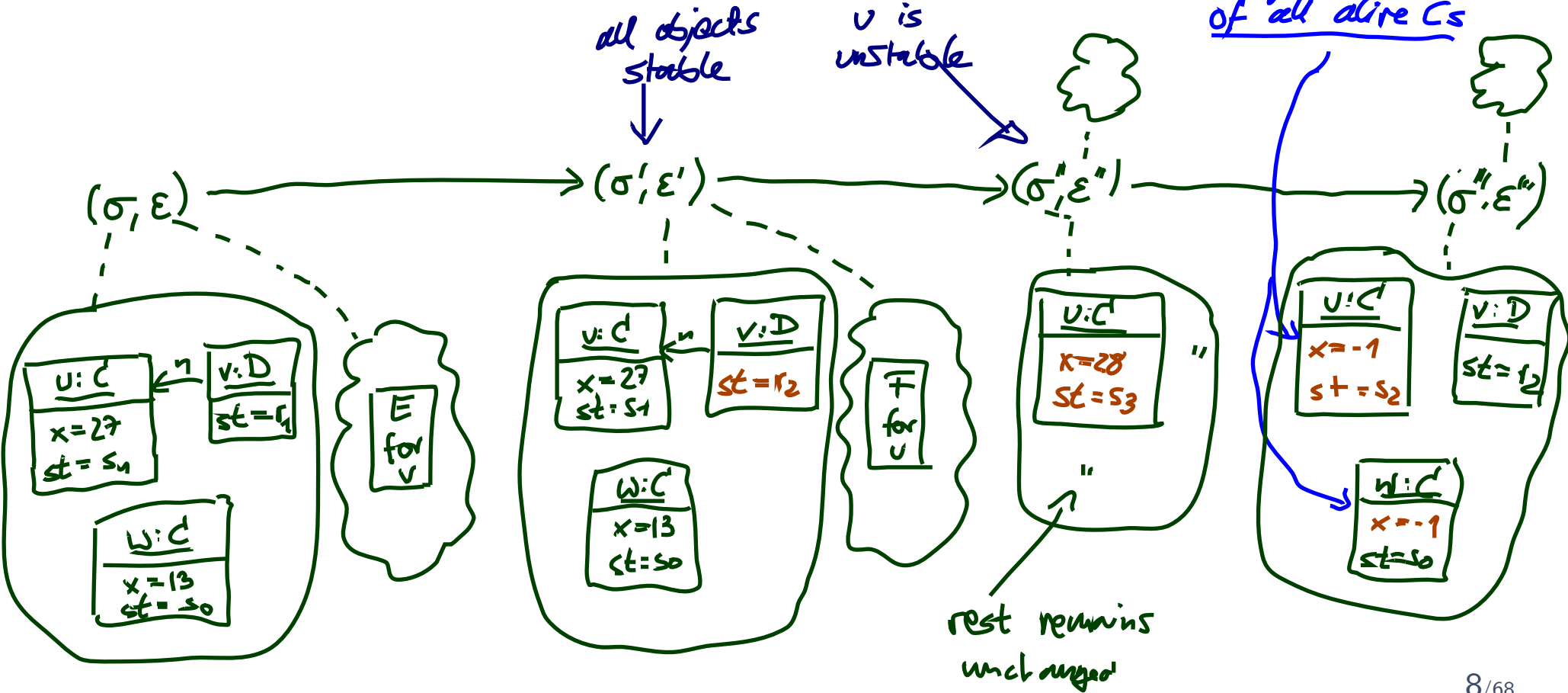
- Event occurrences are detected, dispatched, and then processed by the state machine, one at a time.
- The semantics of event occurrence processing is based on the **run-to-completion assumption**, interpreted as **run-to-completion processing**.
- **Run-to-completion processing** means that an event [...] can only be taken from the pool and dispatched if the processing of the previous [...] is fully completed.
- The processing of a single event occurrence by a state machine is known as a **run-to-completion step**.
- Before commencing on a **run-to-completion step**, a state machine is in a **stable state** configuration with all entry/exit/internal-activities (but not necessarily do-activities) completed.
- The same conditions apply after the **run-to-completion step** is completed.
- Thus, an event occurrence will never be processed [...] in some intermediate and inconsistent situation.
- [IOW,] The **run-to-completion step** is the passage between two state configurations of the state machine.
- The **run-to-completion assumption** simplifies the transition function of the StM, since concurrency conflicts are avoided during the processing of event, allowing the StM to safely complete its **run-to-completion step**.

15.3.12 StateMachine [OMG, 2007b, 563]

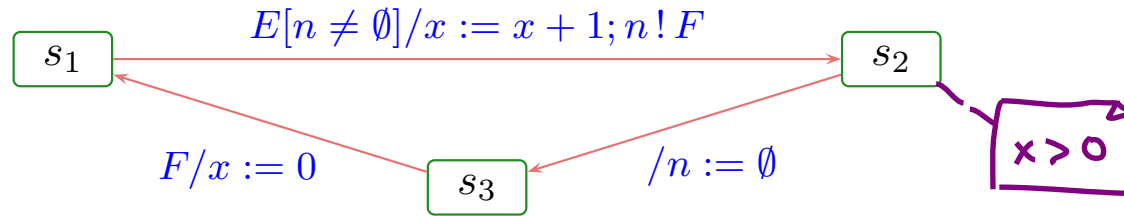
- The order of dequeueing is **not defined**, leaving open the possibility of modeling different priority-based schemes.
- Run-to-completion may be implemented in **various ways**. [...]



our choice of the effect of δ :
 assign -1 to x 's
 of all alive Cs

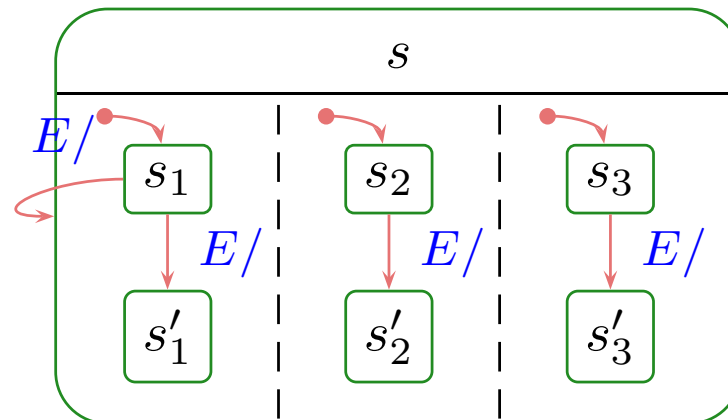


And?



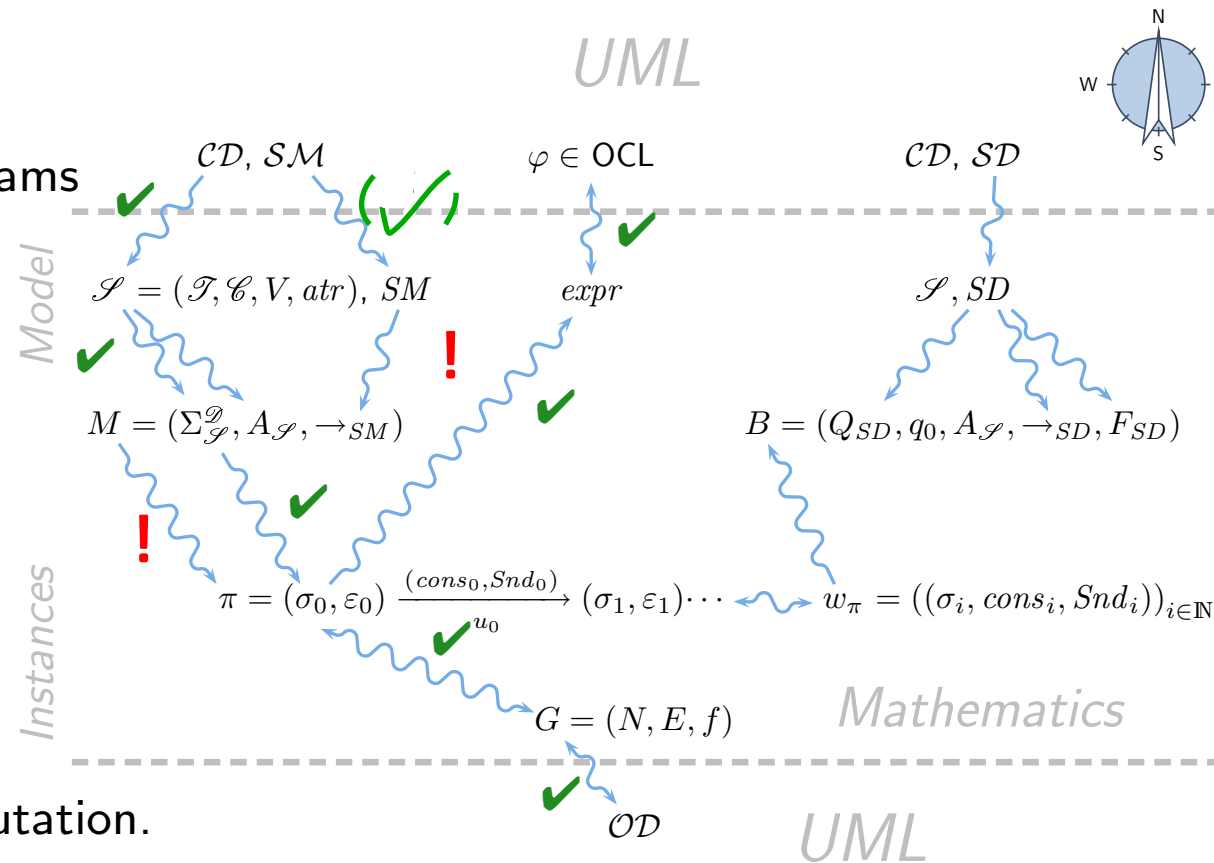
-

- We have to formally define what **event occurrence** is.
- We have to define where events **are stored** – what the event pool is.
- We have to explain how **transitions are chosen** – “matching”.
- We have to explain what the **effect of actions** is – on state and event pool.
- We have to decide on the **granularity** — micro-steps, steps, run-to-completion steps (aka. super-steps)?
- We have to formally define a notion of **stability** and RTC-step **completion**.
- And then: hierarchical state machines.



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.



System Configuration, Ether, Transformer

Ether aka. Event Pool

Definition. Let $\mathcal{S} = (\mathcal{I}, \mathcal{C}, V, atr, \mathcal{E})$ be a signature with signals and \mathcal{D} a structure.

We call a tuple $(Eth, ready, \oplus, \ominus, [\cdot])$ an **ether** over \mathcal{S} and \mathcal{D} if and only if it provides

- a **ready** operation which yields a set of events that are ready for a given object, i.e.

for an event pool \mathcal{E}_m ... and an object identity u ... obtain a set of signal identities (or events)

$$ready : Eth \times \mathcal{D}(\mathcal{C}) \rightarrow 2^{\mathcal{D}(\mathcal{E})}$$

set of \mathcal{E} instances

- a operation to **insert** an event destined for a given object, i.e.

for \mathcal{E}_m , destination, event id

$$\oplus : Eth \times \mathcal{D}(\mathcal{C}) \times \mathcal{D}(\mathcal{E}) \rightarrow Eth$$

obtain a new event pool \mathcal{E}'

- a operation to **remove** an event, i.e.

\mathcal{E}_m event id \mathcal{E}'

$$\ominus : Eth \times \mathcal{D}(\mathcal{E}) \rightarrow Eth$$

- an operation to **clear** the ether for a given object, i.e.

$$[\cdot] : Eth \times \mathcal{D}(\mathcal{C}) \rightarrow Eth.$$

Ether: Examples

$$(Eth, ready, \oplus, \ominus, [\cdot])$$

$$ready: Eth \times \mathcal{D}(C) \rightarrow 2^{\mathcal{D}(E)}$$

- A (single, global, shared, reliable) FIFO queue is an ether:

- $Eth = (\mathcal{D}(C) \times \mathcal{D}(E))^*$

seq. concat. the set of all finite sequences of pairs $(v, e) \in \mathcal{D}(C) \times \mathcal{D}(E)$

id of a signal instance
id of destination object

- $ready((v, e).E, v) = \begin{cases} \{(v, e)\}, & \text{if } v=v \\ \emptyset, & \text{otherwise} \end{cases}$

- $\oplus(E, v, e) = E.(v, e)$

- $\ominus((v, e).E, f) = \begin{cases} E, & \text{if } f=e \\ (v, e).E, & \text{otherwise} \end{cases}$

- $[\cdot]$: remove all (v, e) pairs from given sequence

- One FIFO queue per active object is an ether. [Rhapsody's choice]

(Lossy queue.) (because $\oplus, ready$ are functions)

- One-place buffer.
- Priority queue.
- Multi-queues (one per sender).
- Trivial example: sink, "black hole".

Set of events

• ...

15.3.12 StateMachine [OMG, 2007b, 563]

- The order of dequeuing is **not defined**, leaving open the possibility of modeling different priority-based schemes.
- Run-to-completion may be implemented in **various ways**. [...]

Ether and [OMG, 2007b]

"receiving takes place"

more conceptual; for us: dispatch / discard

The standard distinguishes (among others)

- **SignalEvent** [OMG, 2007b, 450] and **Reception** [OMG, 2007b, 447].

On **SignalEvents**, it says

for us: event

A signal event represents the receipt of an asynchronous signal instance. A signal event may, for example, cause a state machine to trigger a transition. [OMG, 2007b, 449]

[...]

Semantic Variation Points

= messages

The means by which requests are transported to their target depend on the type of requesting action, the target, the properties of the communication medium, and numerous other factors.

In some cases, this is instantaneous and completely reliable while in others it may involve transmission delays of variable duration, loss of requests, reordering, or duplication.

(See also the discussion on page 421.) [OMG, 2007b, 450]

Our **ether** is a general representation of the possible choices. (Γ_{-1} needs relations)

Often seen minimal requirement: order of sending **by one object** is preserved.

~~But: we'll later briefly discuss "discarding" of events.~~

System Configuration

Definition. Let $\mathcal{S}_0 = (\mathcal{I}_0, \mathcal{C}_0, V_0, atr_0, \mathcal{E})$ be a signature with signals, \mathcal{D}_0 a structure of \mathcal{S}_0 , $(Eth, ready, \oplus, \ominus, [\cdot])$ an ether over \mathcal{S}_0 and \mathcal{D}_0 . Furthermore assume there is one core state machine M_C per class $C \in \mathcal{C}$.

A **system configuration** over \mathcal{S}_0 , \mathcal{D}_0 , and Eth is a pair

a type name for each state machine

$$(\sigma, \varepsilon) \in \Sigma_{\mathcal{S}}^{\mathcal{D}} \times Eth$$

where

- $\mathcal{S} = (\mathcal{I}_0 \dot{\cup} \{S_{M_C} \mid C \in \mathcal{C}\}, \mathcal{C}_0,$

$$V_0 \dot{\cup} \{\langle stable : Bool, -, true, \emptyset \rangle\}$$

$$\dot{\cup} \{\langle st_C : S_{M_C}, +, s_0, \emptyset \rangle \mid C \in \mathcal{C}\}$$

$$\dot{\cup} \{\langle params_E : E_{0,1}, +, \emptyset, \emptyset \rangle \mid E \in \mathcal{E}_0\},$$

$$\{C \mapsto atr_0(C)$$

$$\cup \{stable, st_C\} \cup \{params_E \mid E \in \mathcal{E}_0\} \mid C \in \mathcal{C}\}, \mathcal{E}_0)$$

- $\mathcal{D} = \mathcal{D}_0 \dot{\cup} \{S_{M_C} \mapsto S(M_C) \mid C \in \mathcal{C}\},$ and

- $\sigma(u)(r) \cap \mathcal{D}(\mathcal{E}_0) = \emptyset$ for each $u \in \text{dom}(\sigma)$ and $r \in V_0$.

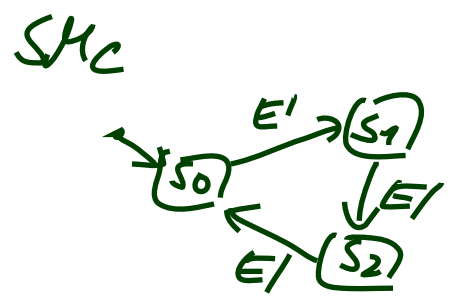
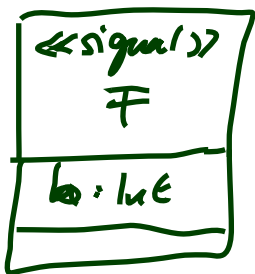
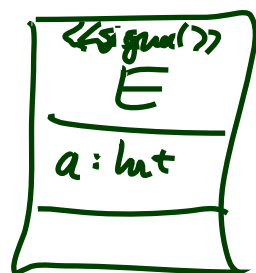
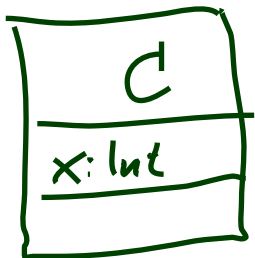
if Bool $\notin \mathcal{I}_0$, then add it and have $\mathcal{D}(Bool) = \mathbb{B}$

initial state of M_C

each object can refer to the signal instances in order to access event attributes

set of states of state machine M_C of C

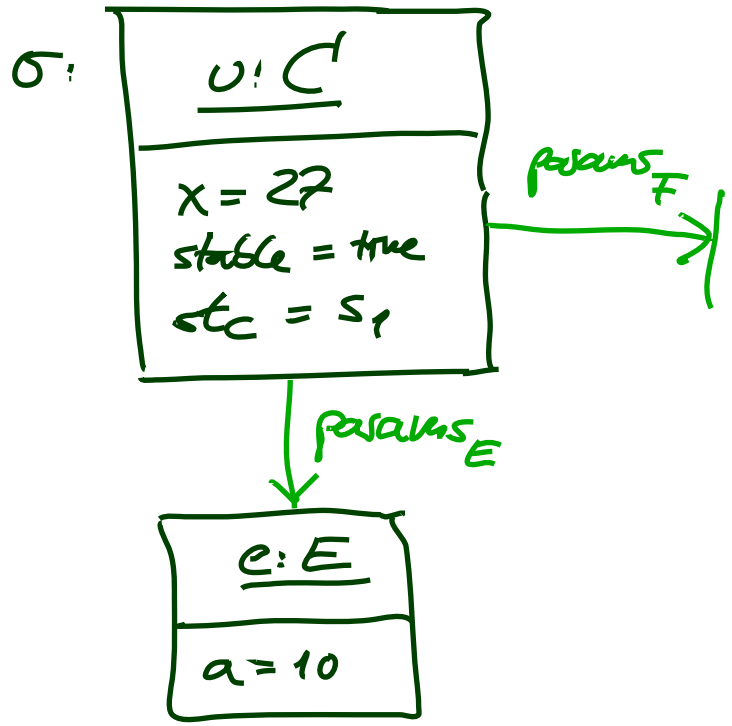
the only links to events are via the params



$$\mathcal{Y}_0 = (\{\text{Int}\}, \{C, E\}, \{a, x\}, \{C \mapsto \{x\}, E \mapsto \{a\}\}, \{E\})$$

$$\mathcal{Y} = (\{\text{Int}, SM_C\}, \{C, E\}, \{a, x, \text{stable}_C: \text{Bool}, \text{st}_C: SM_C\} \cup \{ \text{params}_E: E_{0,1}, \text{params}_F: F_{0,1} \}, \{C \mapsto \{x, \text{stable}_C, \text{st}_C\} \cup \{ \text{params}_E, \text{params}_F \}, E \mapsto \{a\}\}, \{E\})$$

$$\mathcal{D}(SM_C) = \{s_0, s_1, s_2\}$$



System Configuration Step-by-Step

- We start with some signature with signals $\mathcal{S}_0 = (\mathcal{T}_0, \mathcal{C}_0, V_0, atr_0, \mathcal{E})$.
- A **system configuration** is a pair (σ, ε) which comprises a system state σ wrt. \mathcal{S} (not wrt. \mathcal{S}_0).
- Such a **system state** σ wrt. \mathcal{S} provides, for each object $u \in \text{dom}(\sigma)$,
 - values for the **explicit attributes** in V_0 ,
 - values for a number of **implicit attributes**, namely
 - a **stability flag**, i.e. $\sigma(u)(stable)$ is a boolean value,
 - a **current (state machine) state**, i.e. $\sigma(u)(st)$ denotes one of the states of core state machine M_C ,
 - a temporary association to access **event parameters** for each class, i.e. $\sigma(u)(params_E)$ is defined for each $E \in \mathcal{E}$.
- For convenience require: there is **no link to an event** except for $params_E$.

References

References

- [Harel and Gery, 1997] Harel, D. and Gery, E. (1997). Executable object modeling with statecharts. *IEEE Computer*, 30(7):31–42.
- [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.