A UML statecharts semantics with message-passing

Jan Jürjens

Software & Systems Engineering
Informatics, TU Munich
Germany

juerjens@in.tum.de

http://www.jurjens.de/jan

Critical systems development

High quality development of critical systems difficult.

Many systems developed, fielded, used that do not satisfy their criticality requirements.

If human life or substantial commercial assets are concerned, need to develop very carefully.
If systems operate under possibility failure or attack need to exclude possible weaknesses.

Problem: correctness in conflict with cost.
Thorough methods of system design expensive, so not used.

Unified Modeling Language

Unified Modeling Language (UML) offers unprecedented opportunity for high-quality critical systems development feasible in industrial context.

- De-facto standard in industrial modeling, large number of developers trained in UML.
- Compared to previous notations with similar community, relatively precisely defined.

UML: need for formality

Nevertheless: UML semantics given only in prose form.

Ambiguous: bad for tool support or verification.

Need mathematically precise semantics for UML.

Formal UML semantics: towards coherent whole

Lot of work towards formal semantics for UML.

So far mostly only UML diagrams in isolation.

Want precise meaning for whole UML specifications.

Put together diagrams to coherent formal semantics.

UML statecharts semantics with message-passing

First formal semantics for UML statecharts to

- model actions and internal activities explicitly
- provide message-passing between different diagrams
- put whole specification documents on formal basis
- ultimately: completely executable UML specifications.

Also activity diagrams (special case of statecharts).

Abstract State Machines

Formal semantics for large part of UML using
Abstract State Machines (Gurevich).

Transition systems. States: multi-sorted first-order structures.

State: set with function names and function interpretations.

ASM: set of states (incl. initial state) and update rule.

Abstract State Machines: Update rules

Update rules: modify function interpretation:

- $f(\bar{s}) := t$
- **if** g **then** R **else** S
- **seq** R S **endseq**
- **iterate**(R)
- . . .

Iteratively fire update rule, starting with initial state.

Interactive ASM: (A, in, out), rules **Init**(A), **Main**(A)

Message exchange

Component S of $\mathcal{A}$ sends message
$msg = op(exp_1, \ldots, exp_n) \in \mathbf{Events}$ to component R:

- S places $R.msg$ into outQu_S.
- $\mathsf{Sched}_{\mathcal{A}}$ distributes messages from out-queues to in-queues:

  $R.msg$ removed from outQu_S, msg added to inQu_R.
- R removes msg from in-queue and processes content.

For operation calls, keep track of sender for return signals.

Actions

Action: expressions of the following forms:

Call action: call$(op(a_1, \ldots, a_n))$ for an n-ary operation $op \in$ **Operation** and expressions $a_i \in$ **Exp**.

Send action: send$(sig(a_1, \ldots, a_n))$ for an n-ary signal $sig \in$ **Signal** and argument $a_i \in$ **Exp**.

Return action: send$(\text{return}_{op}(a))$ for an operation $op \in$ **Operation** with return value $a \in$ **Exp**.

Assignment: $att := exp$ where $att \in$ **Attribute** is an attribute and $exp \in$ **Exp** an expression.

Void action: nil

Activities

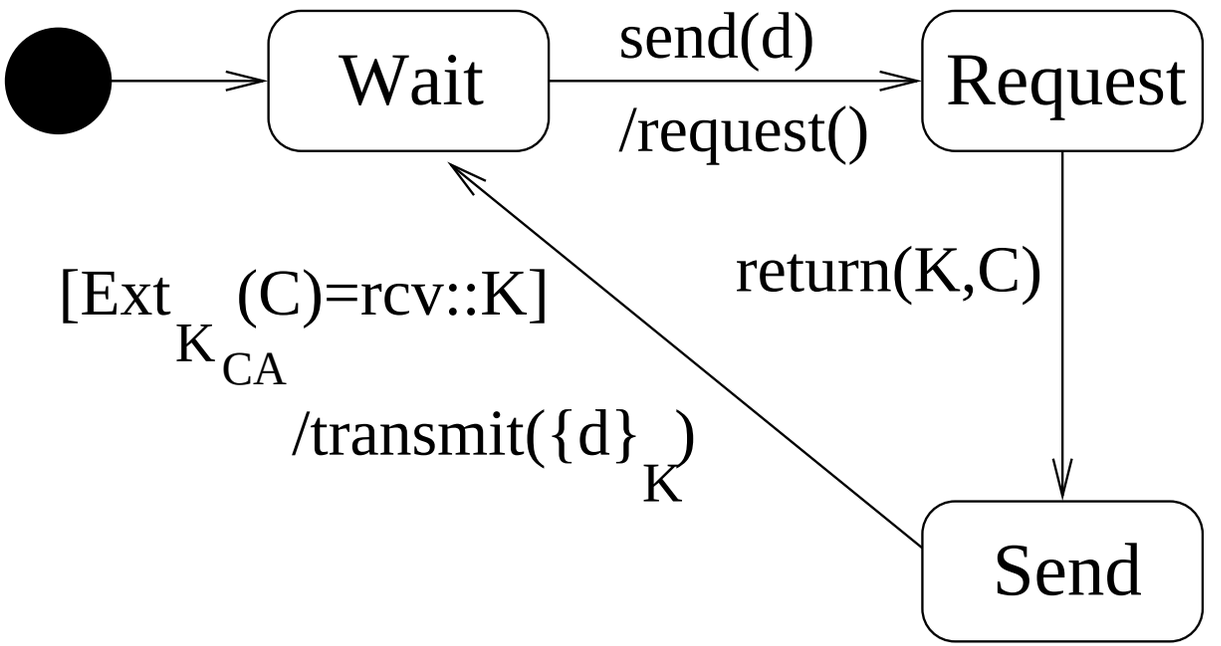
Activity: represents activities used in UML specification.

Assume contains $nil \in$ **Activity** (absence of activity).

For every $actv \in$ **Activity** exists ASM rule **ActvRule**$(actv)$.

nil has ASM rule that sets finished to $true$.

Statecharts



Simplifications

For readability:

- No deferred events.
- No history states.
- No boundary-crossing transitions, from composite states only completion transitions.

(can be extended to general case).

Statechart semantics

State machines process one event at a time (run-to-completion step).

Statechart D defines interactive ASM $[\![D]\!]^{SC}$.

Build on ASM semantics for UML statecharts in Börger, Cavarra, Riccobene 2000.

Extend with actions, activities, message passing.

Semantics of actions

Call/send action:

$$\mathbf{ActionRule}(\mathsf{call}(e)) \equiv \mathsf{tooutQu}_{\mathcal{A}}(\{\!|\, e \,|\!\}\,)$$
$$\mathbf{ActionRule}(\mathsf{send}(e)) \equiv \mathsf{tooutQu}_{\mathcal{A}}(\{\!|\, e \,|\!\}\,)$$

Assignment:

$$\mathbf{ActionRule}(att := exp) \equiv att := exp$$

Void action:

$$\mathbf{ActionRule}(nil) \equiv \mathbf{skip}$$

Example

case currState **of**

$\{\mathsf{Initial}_{Sndr}\}$: **do** currState $:= \{Wait\}$

$\{Wait\}$: **do choose** e **with** $e \in \mathsf{inQu}_{[\![Sndr]\!]SC}$ **do**

do $-$ **in** $-$ **parallel**

$\mathsf{inQu}_{[\![Sndr]\!]SC} := \mathsf{inQu}_{[\![Sndr]\!]SC} \setminus \{\!\{ e \}\!\}$

if msgnm$(e) = send$ **then**

do $-$ **in** $-$ **parallel**

currState $:= \{Send\}$

$d :=$ **Args**(e)

enddo

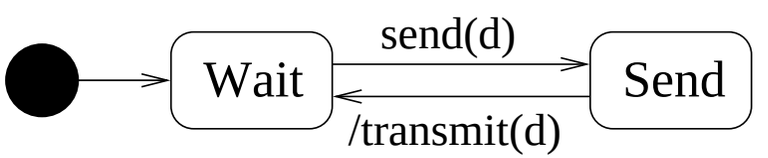
enddo

$\{Send\}$: **do do** $-$ **in** $-$ **parallel**

currState $:= \{Wait\}$

$\mathsf{tooutQu}_{[\![Sndr]\!]SC}(\{\!\{ transmit(d) \}\!\})$

enddo



Conclusion

Significant step towards formal modeling of complete UML specifications.

Beyond formal models of single diagrams in isolation.

First semantics to explicitly model actions, internal activities, operations with their parameters, message-passing between different diagrams and event dispatching.

First step towards executable UML modeling.

Has been extended to other diagrams (Jürjens 2001).

Applications to security (early afternoon session).

Resources

Slides, papers etc.:

http://www.jurjens.de/jan

Thanks for your attention !