Formal Semantics for Interacting UML subsystems

Jan Jürjens

Software & Systems Engineering
Computer science, Munich University of Technology
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.

Formal Semantics for Interacting UML subsystems

Provide formal semantics for (fragment of) UML subsystems including contained diagrams.

Provides events dispatching, action handling.

Interaction via message-passing.

Allows to compose subsystems.

Give consistency conditions for diagrams in subsystem.

Behavioural equivalence, refinement.

Towards executable UML specifications.

Abstract State Machines

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

Transition systems.

States: multi-sorted first-order structures
(set with function names and function interpretations).

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

Abstract State Machines: Update rules

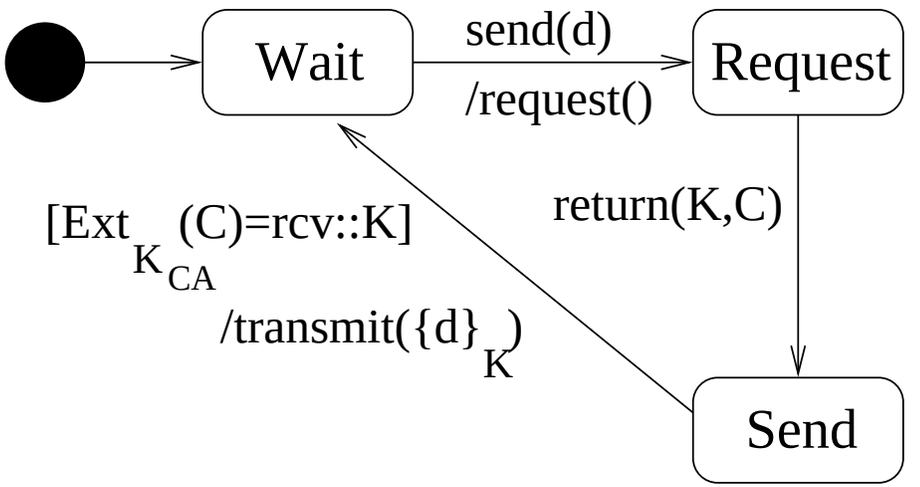
Update rules: modify function interpretation:

- $f(\bar{s}) := t$
- **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)

Statecharts



States with entry/exit actions, substates, internal activities.

Here: simplified.

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.

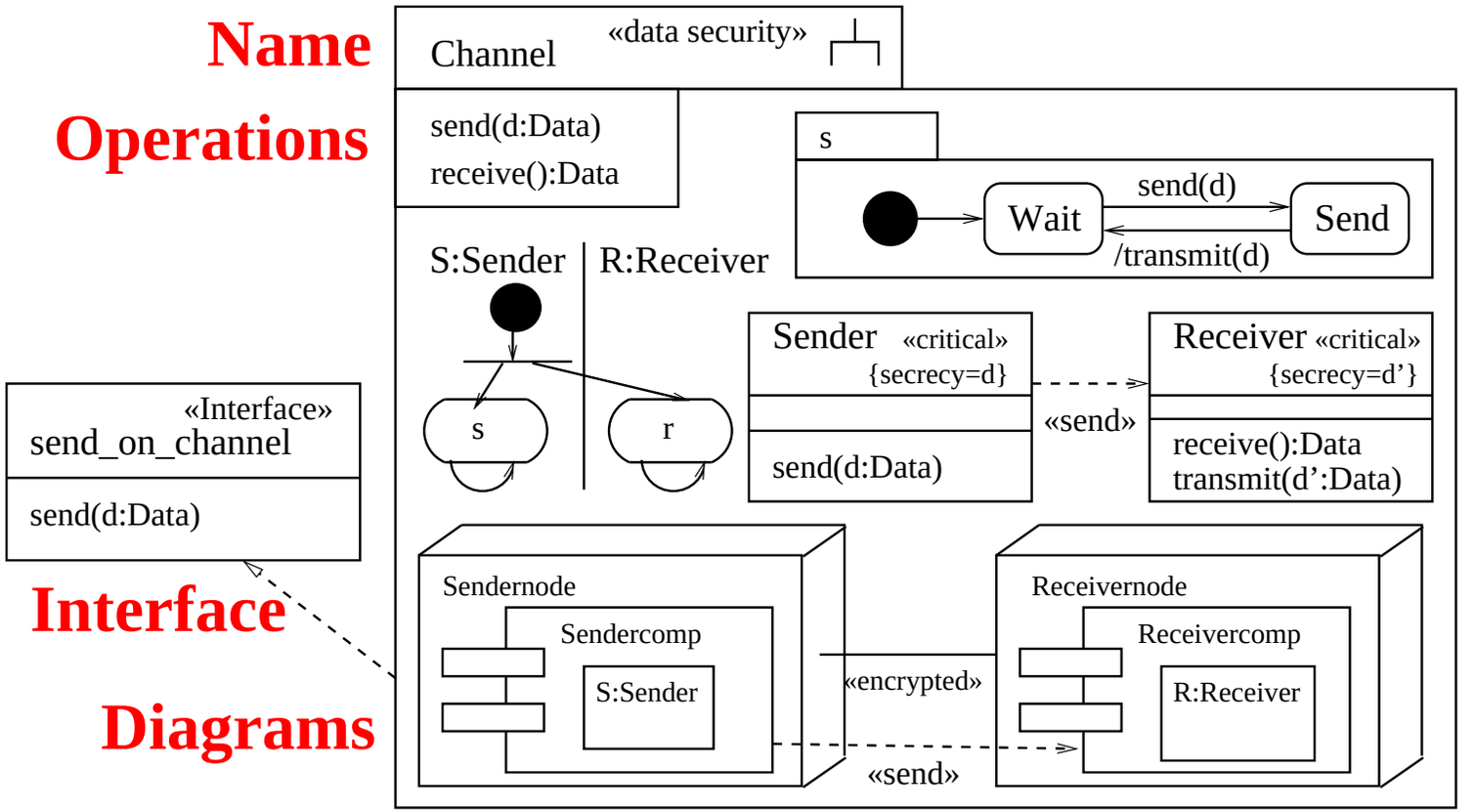
Extend with actions, activities, message passing.

Includes activity diagrams.

Statechart semantics (II)

$Rule$ **Init**(D)
do − in − parallel
  $\mathsf{inQu}_{\mathsf{Object}_D} := \emptyset$ $\mathsf{outQu}_{\mathsf{Object}_D} := \emptyset$
  $\mathsf{currState} := \{\mathsf{Initial}_D\}$ $\mathsf{finished}_{\mathsf{Initial}_D} := \mathit{false}$
enddo
$Rule$ **Main**(D)
seq **if** Completed $\neq \emptyset$ **then** eventExecution(ComplEv)
  else choose e **with** $e \in \mathsf{inQu}_O$ **do**
    seq $\mathsf{inQu}_O := \mathsf{inQu}_O \setminus \{\!\{ e \}\!\}$
      if $e = op_{sender}[args] \in$ **Operation**
        then seq $e := op[args]$
          $\mathsf{trigsusy}(e) := sender$ **endseq**
      eventExecution(e)
    endseq
  loop S **through set** currState
    seq $\mathsf{finished}_S := \mathit{false}$
      ActvRule(internal(S))
    endseq
endseq

Subsystems



May be used to organize model elements into groups.

Here simplified fragement: containing static structure diagram, activity diagram, set of statechart diagrams.

Formal semantics

Consistent subsystem $\mathcal{S}$ gives ASM $[\![\mathcal{S}]\!]$:

$Rule$ **Init**$(\mathcal{S})$
do − in − parallel
  Init(Ad($\mathcal{S}$)) $\mathsf{inQu}_{\mathcal{S}} := \emptyset$ $\mathsf{outQu}_{\mathcal{S}} := \emptyset$
  forall S **with** $S \in \mathsf{Acts}(\mathcal{S})$ **do** **BDInitialize**$_{\mathcal{S}}(S)$
enddo
$Rule$ **Main**$(\mathcal{S})$
seq
  forall S **with** $S \in \mathsf{Acts}(\mathcal{S})$ **do**
    $\mathsf{inQu}_S := \mathsf{inQu}_S \uplus \{\!\{\mathbf{tail}(e) : e \in (\mathsf{inQu}_{\mathcal{S}} \setminus \mathsf{Msgs}(\mathcal{S})) \wedge \mathbf{head}(e) = S\}\!\}$
  $\mathsf{inQu}_{\mathcal{S}} := \emptyset$
  Main(Ad($\mathcal{S}$))
  forall S **with** $S \in \mathsf{Acts}(\mathcal{S})$ **do**
    $\mathsf{inQu}_S := \mathsf{inQu}_S \uplus \biguplus_{T \in \mathsf{Acts}(\mathcal{S})} \{\!\{\mathbf{tail}(e) : e \in \mathsf{outQu}_T \wedge \mathbf{head}(e) = S\}\!\}$
  $\mathsf{outQu}_{\mathcal{S}} := \mathsf{outQu}_{\mathcal{S}} \uplus \biguplus_{T \in \mathsf{Acts}(\mathcal{S})} \{\!\{\mathbf{tail}(e) : e \in \mathsf{outQu}_T \wedge \mathbf{head}(e) = \mathcal{S}\}\!\}$
  forall S **with** $S \in \mathsf{Acts}(\mathcal{S})$ **do** $\mathsf{outQu}_S := \emptyset$
endseq

Timed behaviour

For system $\mathcal{S}$ and multi-sets $\vec{I}_1, \ldots, \vec{I}_n$, timed behaviour $[\![\mathcal{S}]\!]^t(\vec{I}_1, \ldots, \vec{I}_n)$ is set of possible contents of $\mathsf{outlist}(\mathcal{S})$ after execution of:

$Rule$ $\mathbf{tSys_A}(\mathcal{S})$
seq $\mathsf{outlist}(\mathcal{S}) := \emptyset$
  $\mathbf{Init}(\mathcal{S})$
  loop i **through list** $[1 \ldots n]$
    seq $\mathsf{inQu}_{\mathcal{S}} := \mathsf{inQu}_{\mathcal{S}} \uplus \vec{I}_i$
      $\mathbf{Main}(\mathcal{S})$
      $\mathsf{outlist}(\mathcal{S}) := \mathsf{outlist}(\mathcal{S}).\mathsf{outQu}_{\mathcal{S}}$
      $\mathsf{outQu}_{\mathcal{S}} := \emptyset$
    endseq
endseq

Untimed behaviour

Untimed behaviour of system $\mathcal{S}$: For multi-set I, define $[\![\mathcal{S}]\!]^u(I)$ to be fix-point of the content of $\mathsf{outQu}_{\mathcal{S}}$ after firing the rule

$$\mathbf{seq}\ \ \mathbf{Init}(\mathcal{S})\ \ \mathsf{inQu}_{\mathcal{S}} := I\ \ \mathbf{endseq}$$

and iterating the rule $\mathbf{Main}(\mathcal{S})$.

Fix-point exists, because $\mathcal{S}$ only adds elements to $\mathsf{outQu}_{\mathcal{S}}$: take directed union of contents of $\mathsf{outQu}_{\mathcal{S}}$ after each execution of $\mathbf{Main}(\mathcal{S})$.

Refinement

Definition. $\mathcal{S}'$ timed (resp. untimed) *$\mathcal{M}$-refinement* of $\mathcal{S}$ if I and IIa (resp. I and IIb) hold:

I $\mathsf{Msgs}(\mathcal{S})\cap\mathcal{M}\subseteq\mathsf{Msgs}(\mathcal{S}')\cap\mathcal{M}$

IIa for all event multi-sets $I_1,\ldots,I_n$ with $\mathbf{msgnm}(e)\in\mathcal{M}$ for each $e\in\bigcup_{i=1,\ldots,n}I_i$, have $[\![\mathcal{S}']\!]^t(I_1,\ldots,I_n)\curvearrowright\mathcal{M}\subseteq[\![\mathcal{S}]\!]^t(I_1,\ldots,I_n)\curvearrowright\mathcal{M}$

IIb for each event multi-set I with $\mathbf{msgnm}(e)\in\mathcal{M}$ for each $e\in I$, have $[\![\mathcal{S}']\!]^u(I)\curvearrowright\mathcal{M}\subseteq[\![\mathcal{S}]\!]^u(I)\curvearrowright\mathcal{M}$ where

$$S\curvearrowright\mathcal{M}\stackrel{\text{def}}{=}\{M\setminus\{e\in\mathbf{Events}:\mathbf{msgnm}(e)\in\mathcal{M}\}:M\in S\}.$$

Refinement: Properties

Reflexive, transitive, preserves all $\mathcal{M}$-safety properties
(sets of sequences of event multi-sets with names in $\mathcal{M}$).

Timed refinement preserved by substitution:

Theorem. If $\mathcal{S}'_i$ timed refinement of $\mathcal{S}_i$ (each i) then
for any parameterized subsystem $\mathcal{S}(\mathcal{Y}_1, \ldots, \mathcal{Y}_n)$,
$\mathcal{S}(\mathcal{S}'_1, \ldots, \mathcal{S}'_n)$ is a timed refinement of $\mathcal{S}(\mathcal{S}_1, \ldots, \mathcal{S}_n)$.

Not for untimed refinements.

Equivalence

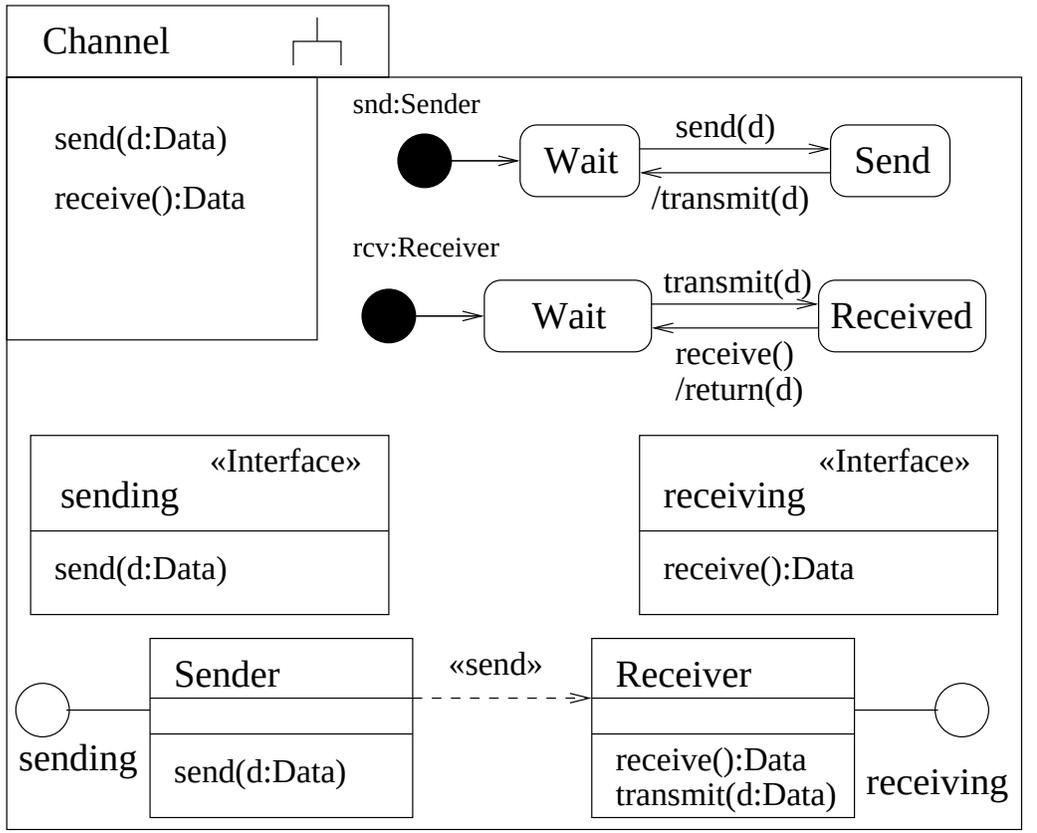
Definition. Subsystem specifications $\mathcal{S}$ and $\mathcal{S}'$ timed (untimed) behaviourally equivalent if

- $\mathrm{Msgs}(\mathcal{S}) = \mathrm{Msgs}(\mathcal{S}')$,
- $\mathcal{S}$ timed (untimed) $\mathrm{Msgs}(\mathcal{S})$-refinement of $\mathcal{S}'$, and
- $\mathcal{S}'$ timed (untimed) $\mathrm{Msgs}(\mathcal{S})$-refinement of $\mathcal{S}$.

Congruence wrt. composition by subsystem formation.

Check if two subsystem specifications describe same behaviour.

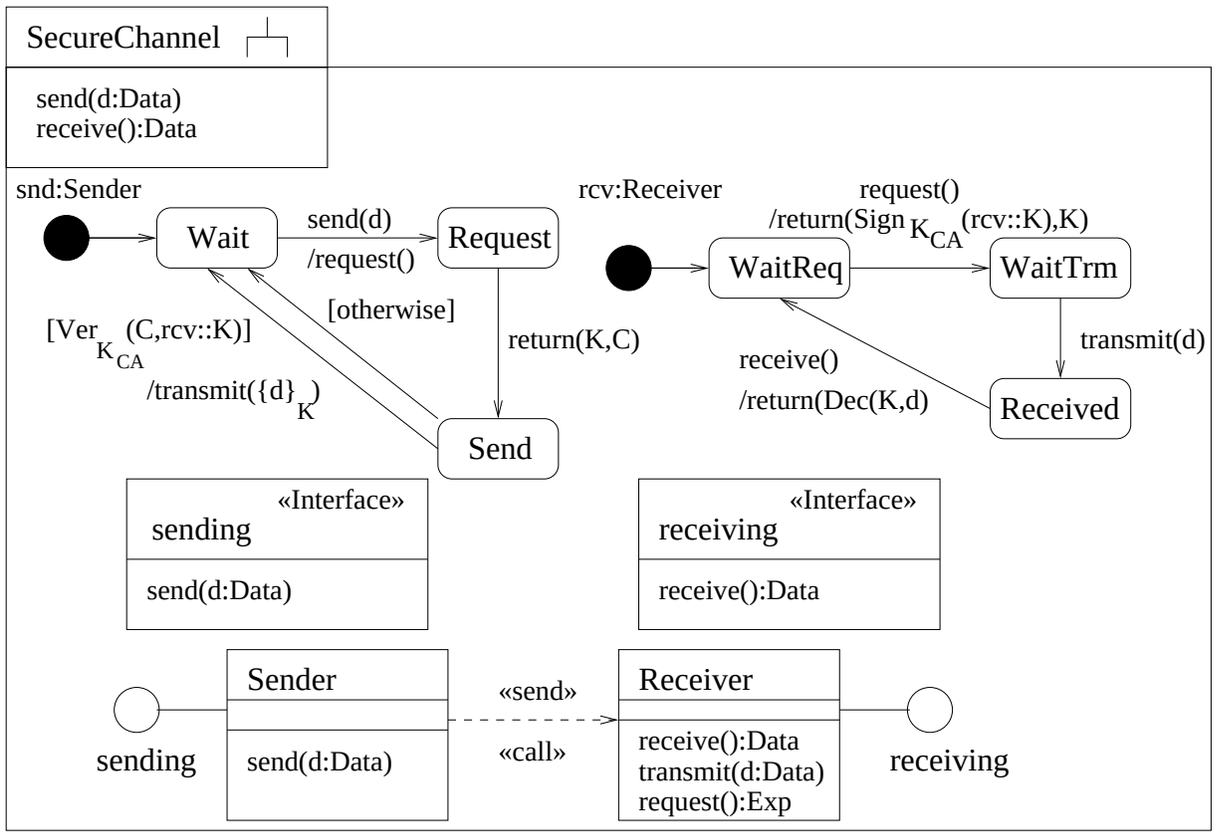
Example: Secure channel establishment



Assume physical layer insecure.

Refinement: secure communication using cryptography.

Example (II)



Sender encrypts data under public key received from receiver.

Theorem. $\mathcal{S}'$ untimed $\{\mathsf{send}, \mathsf{receive}, \mathsf{return}_{\mathsf{receive}}\}$-refinement of $\mathcal{S}$.

Conclusion

Formal semantics for UML subsystems including contained diagrams.

Beyond formal models of single diagrams in isolation.

Towards executing complete UML specifications.

Applications to security (Jürjens 2001).

http://www.jurjens.de/jan