

# Formal Semantics for Interacting UML subsystems

Jan Jürjens

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



[juerjens@in.tum.de](mailto: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:

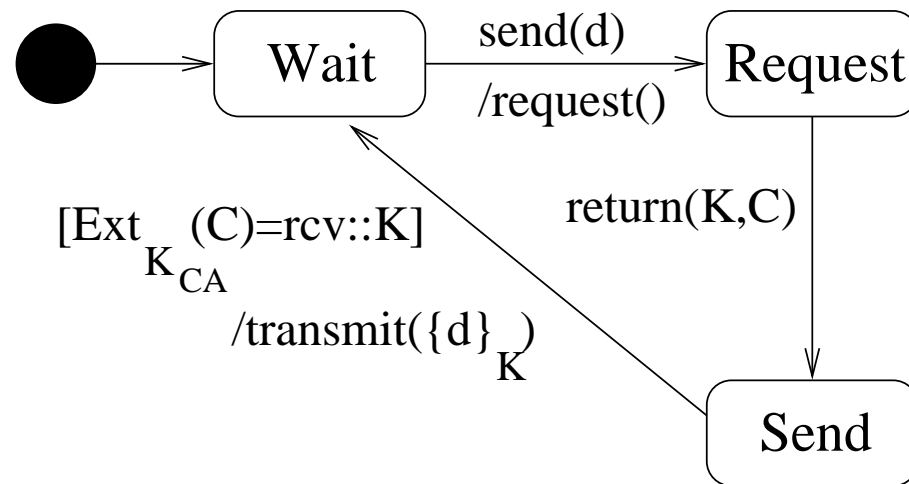
- $f(\bar{s}) := t$
- **if**  $g$  **then**  $R$  **else**  $S$
- **seq**  $R$   $S$  **endseq**
- **iterate**( $R$ )
- . . .

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  $\llbracket D \rrbracket^{SC}$ .

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

Extend with actions, activities, message passing.

Includes activity diagrams.

## Statechart semantics (II)

---

*Rule Init(D)*

**do – in – parallel**

inQu<sub>Object<sub>D</sub></sub> :=  $\emptyset$

outQu<sub>Object<sub>D</sub></sub> :=  $\emptyset$

currState := {Initial<sub>D</sub>}

finished<sub>Initial<sub>D</sub></sub> := *false*

**enddo**

*Rule Main(D)*

**seq** if Completed  $\neq \emptyset$  then eventExecution(ComplEv)

else choose *e* with  $e \in \text{inQu}_O$  do

seq inQu<sub>O</sub> := inQu<sub>O</sub> \ {*e*}

if  $e = op_{sender}[args] \in \mathbf{Operation}$

then seq  $e := op[args]$

trigsusy(*e*) := *sender* endseq

eventExecution(*e*)

endseq

**loop** *S* through set currState

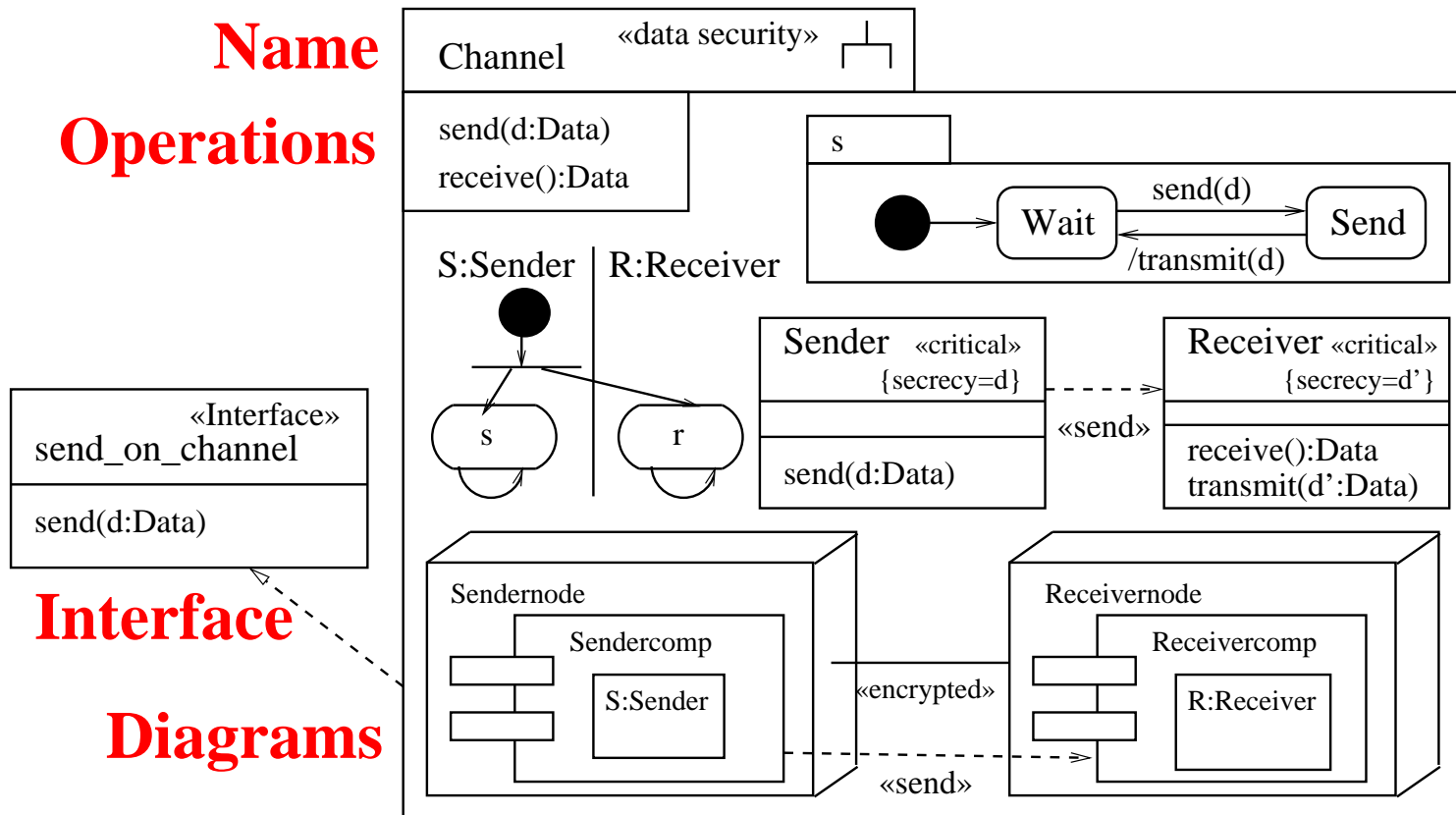
seq finished<sub>*S*</sub> := *false*

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  $\llbracket \mathcal{S} \rrbracket$ :

*Rule*  $\text{Init}(\mathcal{S})$

do – in – parallel

$\text{Init}(\text{Ad}(\mathcal{S}))$      $\text{inQu}_{\mathcal{S}} := \emptyset$      $\text{outQu}_{\mathcal{S}} := \emptyset$

    forall  $S$  with  $S \in \text{Acts}(\mathcal{S})$  do     $\text{BDInitialize}_{\mathcal{S}}(S)$

enddo

*Rule*  $\text{Main}(\mathcal{S})$

seq

    forall  $S$  with  $S \in \text{Acts}(\mathcal{S})$  do

$\text{inQu}_{\mathcal{S}} := \text{inQu}_{\mathcal{S}} \uplus \{ \{ \text{tail}(e) : e \in (\text{inQu}_{\mathcal{S}} \setminus \text{Msgs}(\mathcal{S})) \wedge \text{head}(e) = S \} \}$

$\text{inQu}_{\mathcal{S}} := \emptyset$

$\text{Main}(\text{Ad}(\mathcal{S}))$

    forall  $S$  with  $S \in \text{Acts}(\mathcal{S})$  do

$\text{inQu}_{\mathcal{S}} := \text{inQu}_{\mathcal{S}} \uplus \biguplus_{T \in \text{Acts}(\mathcal{S})} \{ \{ \text{tail}(e) : e \in \text{outQu}_T \wedge \text{head}(e) = S \} \}$

$\text{outQu}_{\mathcal{S}} := \text{outQu}_{\mathcal{S}} \uplus \biguplus_{T \in \text{Acts}(\mathcal{S})} \{ \{ \text{tail}(e) : e \in \text{outQu}_T \wedge \text{head}(e) = \mathcal{S} \} \}$

    forall  $S$  with  $S \in \text{Acts}(\mathcal{S})$  do     $\text{outQu}_{\mathcal{S}} := \emptyset$

endseq

# Timed behaviour

---

For system  $\mathcal{S}$  and multi-sets  $\vec{I}_1, \dots, \vec{I}_n$ , timed behaviour  $\llbracket \mathcal{S} \rrbracket^t(\vec{I}_1, \dots, \vec{I}_n)$  is set of possible contents of  $\text{outlist}(\mathcal{S})$  after execution of:

```
Rule tSysA( $\mathcal{S}$ )  
seq outlist( $\mathcal{S}$ ) :=  $\emptyset$   
  Init( $\mathcal{S}$ )  
  loop  $i$  through list [1 . . .  $n$ ]  
    seq inQu $\mathcal{S}$  := inQu $\mathcal{S}$   $\uplus$   $\vec{I}_i$   
      Main( $\mathcal{S}$ )  
      outlist( $\mathcal{S}$ ) := outlist( $\mathcal{S}$ ).outQu $\mathcal{S}$   
      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  $\text{outQu}_{\mathcal{S}}$  after firing the rule

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

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

Fix-point exists, because  $\mathcal{S}$  only **adds** elements to  $\text{outQu}_{\mathcal{S}}$ : take directed union of contents of  $\text{outQu}_{\mathcal{S}}$  after each execution of **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:

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

**IIa** for all event multi-sets  $I_1, \dots, I_n$  with  $\text{msgnm}(e) \in \mathcal{M}$  for each  $e \in \bigcup_{i=1, \dots, n} I_i$ , have

$$\llbracket \mathcal{S}' \rrbracket^t(I_1, \dots, I_n) \curvearrowright \mathcal{M} \subseteq \llbracket \mathcal{S} \rrbracket^t(I_1, \dots, I_n) \curvearrowright \mathcal{M}$$

**IIb** for each event multi-set  $I$  with  $\text{msgnm}(e) \in \mathcal{M}$  for each  $e \in I$ , have  $\llbracket \mathcal{S}' \rrbracket^u(I) \curvearrowright \mathcal{M} \subseteq \llbracket \mathcal{S} \rrbracket^u(I) \curvearrowright \mathcal{M}$  where

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

## 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  $\mathcal{S}(\mathcal{Y}_1, \dots, \mathcal{Y}_n)$ ,  $\mathcal{S}(\mathcal{S}'_1, \dots, \mathcal{S}'_n)$  is a timed refinement of  $\mathcal{S}(\mathcal{S}_1, \dots, \mathcal{S}_n)$ .

Not for untimed refinements.

# Equivalence

---

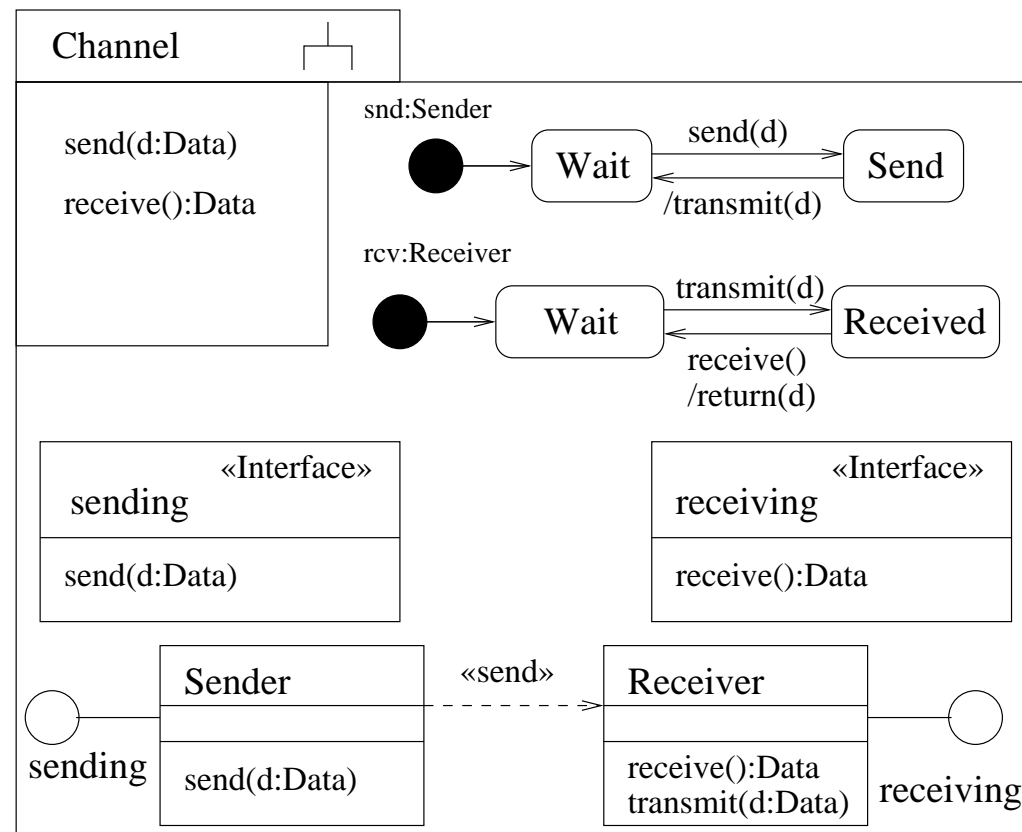
**Definition.** Subsystem specifications  $\mathcal{S}$  and  $\mathcal{S}'$  **timed (untimed) behaviourally equivalent** if

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

Congruence wrt. composition by subsystem formation.

Check if two subsystem specifications describe same behaviour.

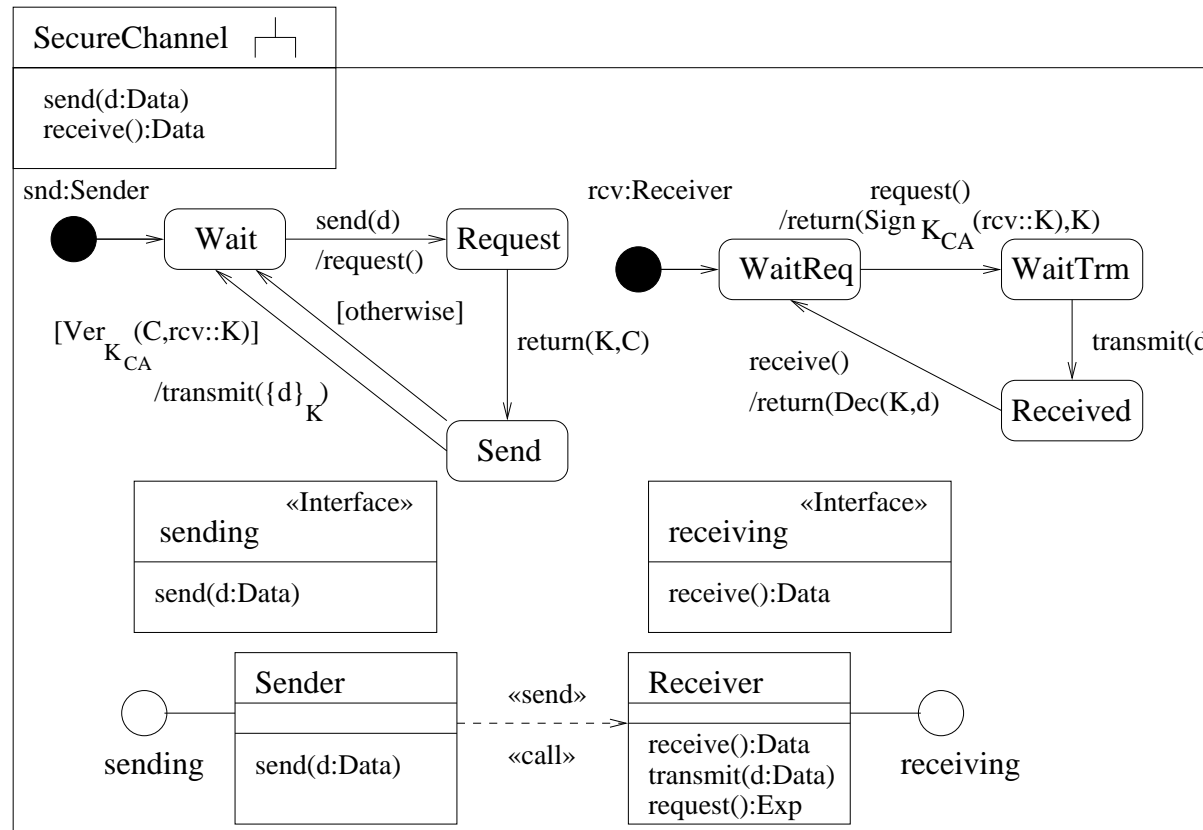
# Example: Secure channel establishment



Assume physical layer insecure.

Refinement: secure communication using cryptography.

# Example (II)



Sender encrypts data under public key received from receiver.

**Theorem.**  $\mathcal{S}'$  untimed  $\{\text{send}, \text{receive}, \text{return}_{\text{receive}}\}$ -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>