

An ASM semantics for the communication layer in Rational Rose Real-Time

Stefan Leue

Alin Ştefănescu

speaking

Software Engineering Group
University of Konstanz

7–November–2005

People:

- Prof. Stefan Leue
- Wei Wei
- Alin Ștefănescu

Project IMCOS:

- “incomplete but fully automated methods for the analysis and verification of concurrent, object-oriented software systems”
- **buffer-boundedness** scalable test for **RoseRT models** and other communicating machines (stage: **mature**)
- **model checking** RoseRT models (stage: **in progress**)

Goal: formal analysis of models as implemented by a **tool**
(Rational Rose Real Time)

- **Formal semantics** of model behavior is **essential**
 - **tuned** analysis algorithms
 - tool documentation also sometimes too informal
 - not enough UML-RT formalization
- **Executable semantics**
 - Abstract State Machines (tool support)
 - previous ASM modeling for UML parts available

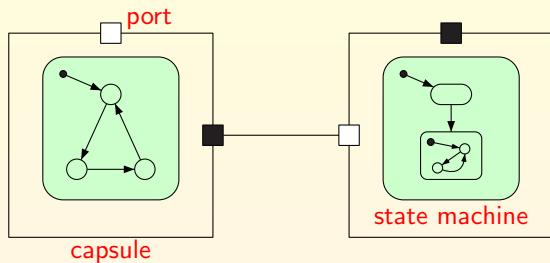
ROOM (1994) & UML (1997) \rightsquigarrow UML-RT (1998)

UML-RT and Rational Rose Real-Time

ROOM (1994) & UML (1997) \rightsquigarrow **UML-RT** (1998)
supported by Rational Rose Real-Time

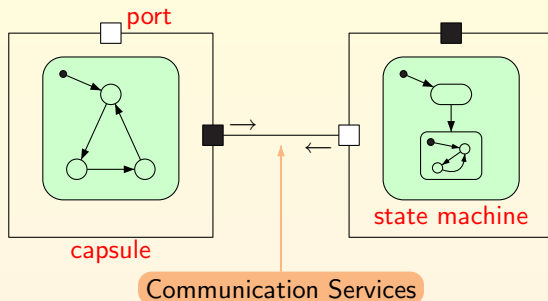
UML-RT and Rational Rose Real-Time

ROOM (1994) & UML (1997) \rightsquigarrow **UML-RT** (1998)
supported by Rational Rose Real-Time



UML-RT and Rational Rose Real-Time

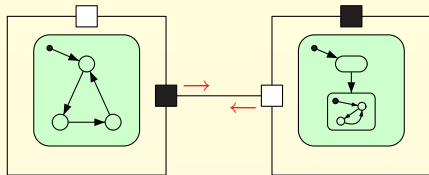
ROOM (1994) & UML (1997) \rightsquigarrow **UML-RT** (1998)
supported by Rational Rose Real-Time



Goal:

- understand and formalize the communication services in RoseRT with Java as underlying detail and target language

Communication layer specified by UML-RT

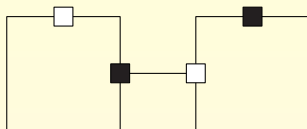


all communication exclusively via **asynchronous** message passing

- p2p connections via **ports** (sole public interface) regulated by binary **protocols**
- event-driven paradigm
- **priority**-based message dispatching
- run-to-completion behavior
→ messages arriving while capsule busy are **queued**

- We choose **Abstract State Machines (ASM)** introduced by Gurevich as semantic envelope for our formalization.
- ASMs are transitions systems with
 - **states** described by **sets with relations and functions**
 - initial state
 - **transitions** described by **update rules** controlling the modification of functions
- **Multi-agents ASMs** (used for our concurrent setting)
 - a set of (sequential) agents
 - each executing a program consisting of ASM rules.

UML-RT capsule signature in ASM



static domain CAPSULE

static interface: CAPSULE $\rightarrow \mathcal{P}(\text{PORT})$

static stateMachine: CAPSULE $\rightarrow \text{STATEMACHINE}$

currMessage: CAPSULE $\rightarrow \text{MESSAGE}$

static domain PORT

static capsule: PORT $\rightarrow \text{CAPSULE}$

static inSignals: PORT $\rightarrow \mathcal{P}(\text{SIGNAL})$

static outSignals: PORT $\rightarrow \mathcal{P}(\text{SIGNAL})$

static conjugate: PORT $\rightarrow \text{PORT}$

constraint $\forall p \in \text{PORT} :$

$p.\text{conjugate.conjugate} = p \wedge p.\text{outSignals} \subseteq p.\text{conjugate.inSignals}$

Communication layer implemented by RoseRT

Controller 1

capsules
messages in transit
scheduler

...

Controller N

capsules
messages in transit
scheduler

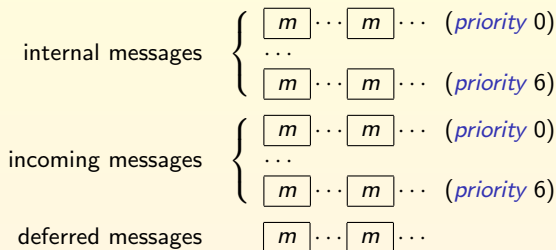
all communication is in the hands of **controllers**

- each controller runs on a separate **thread**
- **one-to-many** relation between controllers and capsules
- **scheduler** iteratively dispatches messages to capsules

Although one thinks of each capsule as a separate 'active agent', we add in **ASM agents only for controllers**, and not for capsules (concurrency is at the level of controllers only).

Topology of a controller

Each controller organizes messages in **global FIFO queues**



static domain CONTROLLER

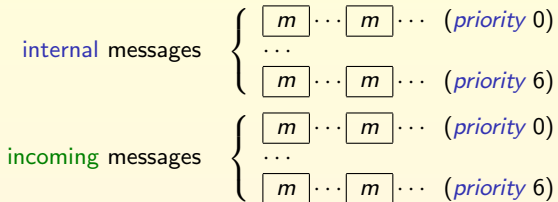
static controller: CAPSULE \rightarrow CONTROLLER

internalQueue: CONTROLLER \times PRIORITY \rightarrow MESSAGE*

incomingQueue: CONTROLLER \times PRIORITY \rightarrow MESSAGE*

deferredQueue: CONTROLLER \times PRIORITY \rightarrow MESSAGE*

Message dispatching (scheduler behavior)



In an infinite loop, the **scheduler** of each controller:

- 1 selects the highest-priority non-empty queue of **incoming** messages
- 2 appends it to the queue of **internal** messages with the *same* priority
- 3 pops the first message in the highest-priority non-empty queue of **internal** messages and
- 4 dispatch it to the corresponding capsule for processing

Scheduler behavior in ASM

internal queues (Q_0, \dots, Q_6) + **incoming** queues (Q_0, \dots, Q_6)

For each agent $a \in \text{AGENT}$ associated to a controller:

Rule *SchedulerMessageDispatching(Self)*

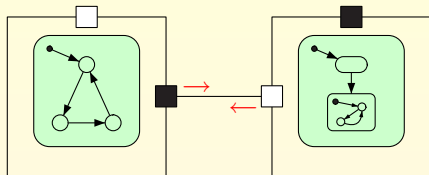
```
let ctrl=Self.controller in
  seq
    appendHighestIncomingQueueToInternal(ctrl)
    let msg=firstHighestInternalMessage(ctrl) in
      messageProcessing(msg,msg.destinationPort.capsule)
      dequeue(msg,ctrl.internalQueue(msg.priority))
```

where e.g.

messageProcessing(msg,capsule) ≡

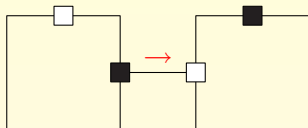
```
seq
  capsule.currMessage := msg
  stateMachineExecution(capsule.stateMachine, msg)
  capsule.currMessage := nil
```

Actions under focus



- **Send** – `port.signal.send(priority)`
- **Invoke** – `port.signal.invoke`
- **Reply** – `port.signal.reply`
- **Defer** – `defer`
- **Recall** – `port.recall(ahead)`
- **Purge** – `port.purge`

Send (asynchronous message)



```
port.signal.send(priority) ≡
```

```
let (destCtrl=port.conjugate.capsule.controller ∧  
    originCtrl=port.capsule.controller ∧  
    msg=⟨port.conjugate, signal, priority, port⟩)  
in  
  if (destCtrl == originCtrl) then  
    enqueue(msg, destCtrl.internalQueue(priority))  
  else  
    enqueue(msg, destCtrl.incomingQueue(priority))
```

Invoke (procedure call)

{Capsule A **invokes** capsule B }: A sends a message to B for processing, **waits** for a reply from B , then continues its execution

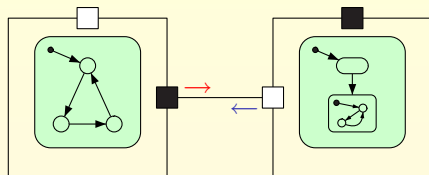
RoseRT restrictions

- invokes are allowed **only inside a controller**
- **no circular** invokes
- invokes cannot be **deferred**

Implementation details

- invoked messages are directly sent to destination capsule for processing (**no queues involved**)
- replies to invokes are specially handled by controllers

currReplyInvoke: CONTROLLER \rightarrow MESSAGE



While processing a message, a capsule may **reply** to the sender

- the reply must be send via the port the original messages was received
- the semantics depending on the original message **msg**:
 - if **msg** is an (ordinary) **send**, then reply is also an ordinary **send** with the **same priority**
 - if **msg** is an **invoke**, then reply will just place an acknowledgment in a dedicated holder on the sender's controller

While processing a message, a capsule may **defer** it, and **recall** it sometime later. Batches of deferred messages may be simply **discarded** (purged), if expired.

- a **global queue** of deferred messages per controller
- **all** defer/recall/purge operations **explicit** in action code
- this approach diverges from e.g. UML 2.0 recommendation:
 - **states** may have lists of **deferrable** events
 - deferred events are **automatically** reconsidered when *“a state is reached where either the event is no longer deferred or where the event triggers a transition.”*

- towards an **executable** semantics for RoseRT (communication)
- adding **timing** facilities
 - timeouts as special messages
- RoseRT **state machines**
 - based on existing ASM semantics for UML state diagrams
- **simulation** and **testing** of the ASM model
 - looking into **Spec Explorer** from Microsoft (and **AsmL**)
- incorporating communication details into a model checker
 - playing with **Bogor** from Kansas University

Critics and **suggestions** are **welcome!**

Appendix

- RoseRT 2 Promela (Spin model checker):
 - only one controller considered
 - bounded queues
- ASM semantics of UML 2.0 diagrams:
 -