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

Stefan Leue    Alin Ștefănescu
speaking

Software Engineering Group
University of Konstanz

7–November–2005
IMCOS project at the University of Konstanz

People:

- Prof. Stefan Leue
- Wei Wei
- Alin Ştefănescu

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)
Motivation

**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
supported by *Rational Rose Real-Time*
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**: $\text{Capsule} \rightarrow \mathcal{P}(\text{Port})$
- **static stateMachine**: $\text{Capsule} \rightarrow \text{StateMachine}$
- **currMessage**: $\text{Capsule} \rightarrow \text{Message}$

**static domain Port**

- **static capsule**: $\text{Port} \rightarrow \text{Capsule}$
- **static inSignals**: $\text{Port} \rightarrow \mathcal{P}(\text{Signal})$
- **static outSignals**: $\text{Port} \rightarrow \mathcal{P}(\text{Signal})$
- **static conjugate**: $\text{Port} \rightarrow \text{Port}$
- **constraint** $\forall p \in \text{Port}$:
  
  $p.\text{conjugate}.\text{conjugate} = p \land p.\text{outSignals} \subseteq p.\text{conjugate}.\text{inSignals}$
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).
Topography of a controller

Each controller organizes messages in global FIFO queues

- **internal messages**
  - \( m \cdots m \cdots \) (priority 0)
  - \( \cdots \)
  - \( m \cdots m \cdots \) (priority 6)

- **incoming messages**
  - \( m \cdots m \cdots \) (priority 0)
  - \( \cdots \)
  - \( m \cdots m \cdots \) (priority 6)

- **deferred messages**
  - \( m \cdots m \cdots \)

**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*
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 \((Q_0, \ldots, Q_6)\) + incoming queues \((Q_0, \ldots, Q_6)\)

For each agent \(a \in AGENT\) associated to a controller:

**Rule SchedulerMessageDispatching** (Self)

```plaintext
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) ≡

```plaintext
seq
  capsule.currMessage := msg
  stateMachineExecution(capsule.stateMachine, msg)
  capsule.currMessage := nil
```
Actions under focus

- Send – \texttt{port.signal.send(priority)}
- Invoke – \texttt{port.signal.invoke}
- Reply – \texttt{port.signal.reply}
- Defer – \texttt{defer}
- Recall – \texttt{port.recall(ahead)}
- Purge – \texttt{port.purge}
Send (asynchronous message)

\[ \text{port.signal.send}(\text{priority}) \equiv \]

\[
\begin{align*}
\text{let} & \quad (\text{destCtrl}=\text{port.conjugate}.\text{capsule}.\text{controller} \land \\
& \quad \text{originCtrl}=\text{port}.\text{capsule}.\text{controller} \land \\
& \quad \text{msg}=\langle \text{port.conjugate}, \text{signal}, \text{priority}, \text{port} \rangle) \\
\text{in} & \\
\text{if} & \quad (\text{destCtrl} == \text{originCtrl}) \text{ then} \\
& \quad \text{enqueue}(\text{msg}, \text{destCtrl}.\text{internalQueue}(\text{priority})) \\
\text{else} & \\
& \quad \text{enqueue}(\text{msg}, \text{destCtrl}.\text{incomingQueue}(\text{priority}))
\end{align*}
\]
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

$\text{currReplyInvoke}: \text{Controller} \rightarrow \text{Message}$
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.”
Next steps

• 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**!