Run-to-Completion Step (1)

\[
\text{RTC}(env, \text{conf}) = \\
\left[ \begin{array}{l}
\text{event} \leftarrow \text{fetch}() \\
\text{step} \leftarrow \text{choose steps}(\text{conf}, \text{event}) \\
\text{if } \text{step} = \emptyset \land \text{event} \in \text{deferred}(\text{conf}) \text{ then } \text{defer}(\text{event}) \text{ fi} \\
\text{for transition } \in \text{step} \text{ do} \\
\text{conf} \leftarrow \text{handleTransition}(env, \text{conf}, \text{transition}) \\
\text{od} \\
\text{if isCall(\text{event}) } \land \text{event} \notin \text{deferred}(\text{conf}) \text{ then } \text{acknowledge}(\text{event}) \text{ fi} \\
\text{conf} \\
\end{array} \right]
\]
Run-to-Completion Step (2)

\[
\text{steps}(env, \text{conf}, \text{event}) \equiv \\
\left[ \begin{array}{c}
\text{transitions} \leftarrow \text{enabled}(env, \text{conf}, \text{event}) \\
\{\text{step} \mid (\text{guard}, \text{step}) \in \text{steps}(\text{conf}, \text{transitions}) \land env \models \text{guard} \} 
\end{array} \right]
\]

\[
\text{steps}(\text{conf}, \text{transitions}) \equiv \\
\left[ \begin{array}{c}
\text{steps} \leftarrow \{(\text{true}, \emptyset)\} \\
\text{for } \text{transition} \in \text{transitions} \text{ do} \\
\text{for } (\text{guard}, \text{step}) \in \text{steps}(\text{conf}, \text{transitions} \setminus \{\text{transition}\}) \text{ do} \\
\text{if } \text{inConflict}(\text{conf}, \text{transition}, \text{step}) \\
\text{then } \text{if } \text{higherPriority}(\text{conf}, \text{transition}, \text{step}) \\
\text{then } \text{guard} \leftarrow \text{guard} \land \neg \text{guard}(	ext{transition}) \text{ fi} \\
\text{else } \text{step} \leftarrow \text{step} \cup \{\text{transition}\} \\
\text{guard} \leftarrow \text{guard} \land \text{guard}(\text{transition}) \text{ fi} \\
\text{steps} \leftarrow \text{steps} \cup \{(\text{guard}, \text{step})\} \text{ od} \\
\text{steps} \leftarrow \text{steps} \cup \{(\text{guard}, \text{step})\} \text{ od} \\
\end{array} \right]
\]
Run-to-Completion Step (3)

\[
\text{handleTransition}(\text{conf, transition}) \equiv \left[ \begin{array}{l}
\text{for state } \in \text{insideOut(exited(transition)) do}
\quad \text{uncomplete(state)}
\quad \text{for timer } \in \text{timers(state)} \text{ do stopTimer(timer) od}
\quad \text{execute(exit(state))}
\quad \text{conf }\leftarrow \text{conf}\setminus \{\text{state}\}
\text{od}
\quad \text{execute(effect(transition))}
\quad \text{for state } \in \text{outsideIn(entered(transition)) do}
\quad \text{execute(entry(state))}
\quad \text{for timer } \in \text{timers(state)} \text{ do startTimer(timer) od}
\quad \text{conf }\leftarrow \text{conf}\cup \{\text{state}\}
\quad \text{complete(conf, state)}
\quad \text{od}
\end{array} \right]
\]

conf
Semantic variation points

• Some semantic variation points have been mentioned before.
  • delays in event pool
  • handling of deferred events
  • entering of composite states without default entry

• Which events are prioritized?
  • completion events only
  • all internal events (completion, time, change)

• Which (additional) timing assumptions?
  • delays in communication
  • time for run-to-completion step
    • zero-time assumption
State machine refinement

- State machines are behaviors and may thus be refined.
Protocol state machines

- Protocol state machines specify which behavioral features of a classifier can be called in which state and under which condition and what effects are expected.
  - particularly useful for object life cycles and ports
  - no effects on transitions, only effect descriptions

![Diagram of Protocol state machines]

**ProtocolTransition**
Protocol state machines

Several operation specifications are combined conjunctively:

context $C::\text{op}()$
pre: \text{inState}(S_1) \text{ and } P_1 
post: Q_1 \text{ and } \text{inState}(S_3)

context $C::\text{op}()$
pre: \text{inState}(S_2) \text{ and } P_2 
post: Q_2 \text{ and } \text{inState}(S_4)

results in

context $C::\text{op}()$
pre: \text{(inState}(S_1) \text{ and } P_1) \text{ or } (\text{inState}(S_2) \text{ and } P_2) 
post: (\text{inState@pre}(S_1) \text{ and } P_1@pre) \text{ implies } (Q_1 \text{ and } \text{inState}(S_3)) 
\quad \text{and} (\text{inState@pre}(S_2) \text{ and } P_2@pre) \text{ implies } (Q_2 \text{ and } \text{inState}(S_4))
How things work together

- Static structure
  - sets the scene for state machine behavior
  - state machines refer to
    - properties
    - behavioral features (operations, receptions)
    - signals
- Interactions
  - may be used to exemplify the communication of state machines
  - refer to event occurrences used in state machines
- OCL
  - may be used to specify guards and pre-/post-conditions
  - refers to actions of state machines (OclMessage)
- Protocols and components
  - state machines may specify protocol roles
Wrap up

• State machines model behaviour
  • object and use case life cycles
  • control automata
  • protocols

• State machines consist of
  • Regions and …
  • … (Pseudo)States (with entry, exit, and do-activities) …
  • connected by Transitions (with triggers, guards, and effects)

• State machines communicate via event pools.

• State machines are executed by run-to-completion steps.
Unified Modeling Language 2

Interactions
A first glimpse

sequence diagram

communication diagram

timing diagram

all three are semantically equivalent
History and predecessors

- Simple sequence diagrams
  - 1990's
    - Message Sequence Charts (MSCs) used in TelCo-industry
    - several OO-methods use sequence diagrams

- Complex sequence diagrams
  - 1996: Complex MSCs introduced in standard MSC96
  - 1999: Life Sequence Charts (LSCs)

- Communication diagrams
  - 1991: used in Booch method
  - 1994: used in Cook/Daniels: Syntropy

- Timing diagrams
  - traditionally used in electrical engineering
  - 1991: used in Booch method
  - 1993: used in early MSCs

- Interaction overview
  - 1996: high-level MSCs (graphs of MSCs as notational alternative)
Usage scenarios

- **Class/object interactions**
  - design or document message exchange between objects
  - express synchronous/asynchronous messages, signals and calls, activation, timing constraints

- **Use case scenarios**
  - illustrate a use case by concrete scenario
  - useful in design/documentation of business processes (i.e. analysis phase and reengineering)

- **Test cases**
  - describe test cases on all abstraction levels

- **Timing specification/documentation**

- **Interaction overview**
  - organize a large number of interactions in a more visual style
  - defined as equivalent to using interaction operators
Syntactical variants

- **Sequence diagram**
  - traditional sequence diagrams + interaction operators
  - focuses on exchanging many messages in complex patterns among few interaction partners

- **Communication diagram**
  - “collaboration diagram” in UML 1.x
  - focuses on exchanging few messages between (many) interaction partners in complex configuration

- **Timing diagram**
  - new in UML 2.0, oscilloscope-type representation, not necessarily metric time
  - focuses on (real) time and coordinated state change of interaction partners over time

- **Interaction overview diagram**
  - looks like restricted activity diagram, but isn’t
  - arrange elementary interactions to highlight their interaction
Main concepts

- **Interaction partner**
- **Lifeline**
- **OccurrenceSpecification** aka. EventOccurrence

```
Client
  callService( self, parameter )

Server
  jobNo
  receiveResult( jobNo )
```

- call
- reply

Asynchronous signal
Message types

- **instantiation Event**
- **termination Event**
- **lost & found Messages** (i.e.: very slow messages)
- **non-instantaneous Message**
Activation

1. external Event
2. activation bar
3. activation suspended
4. warp lines (non-UML)
5. nested activation
Usage: Use case scenarios

- Interaction **participants** are actors and systems rather than classes and objects.
- May be **refined** successively.
- Useful also for specifying high-level non-functional requirements such as response times.
- All kinds of interaction diagrams may be applied, depending on the circumstances.
Usage: Class interactions

- Interaction **participants** are classes and objects rather than actors and systems.

- Again, successive **refinement** may be applied in different styles:
  - break down processing of messages
  - break down structure of interaction participants.

- All kinds of interaction diagrams may be applied, depending on the circumstances.
Usage: Test cases

- Like any other interaction, but with a different intention.

- Typically accompanied by a **tabular description** of purpose, expected parameters and result (similar to use case description).

<table>
<thead>
<tr>
<th>identifier</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>TF-AAA.CIA-4</td>
<td>Check In (automatic) too much luggage</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>test goal</th>
</tr>
</thead>
<tbody>
<tr>
<td>If a passenger has too many pieces of luggage and tries to check in using the check in machine, he should be referred to the check in counter.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>precondition</th>
</tr>
</thead>
<tbody>
<tr>
<td>passenger is booked on respective flight</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>arguments</th>
</tr>
</thead>
<tbody>
<tr>
<td>luggage, bonus mile card, booking data</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>result</th>
</tr>
</thead>
<tbody>
<tr>
<td>passenger is referred to counter</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>postcondition</th>
</tr>
</thead>
<tbody>
<tr>
<td>luggage is not checked in, passenger is checked in</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>remarks, open questions</th>
</tr>
</thead>
<tbody>
<tr>
<td>none</td>
</tr>
</tbody>
</table>
Usage: Timing specification

- For **embedded** and **real-time** systems, it may be important to specify absolute timings and state evolution over time.

- This is not readily expressed in sequence diagrams, much less communication diagrams.

- UML 2.0 introduces **timing diagrams** for this purpose.