EEE499 – Model-driven Development of Real-Time Systems

UML-RT and Papyrus-RT: Advance Behavioural Modeling

ROYAL MILITARY COLLEGE OF CANADA
ELECTRICAL & COMPUTER ENGINEERING

GÉNIE ÉLECTRIQUE ET GÉNIE INFORMATIQUE
COLLÈGE MILITAIRE ROYAL DU CANADA

RMC CMR

Acknowledgement

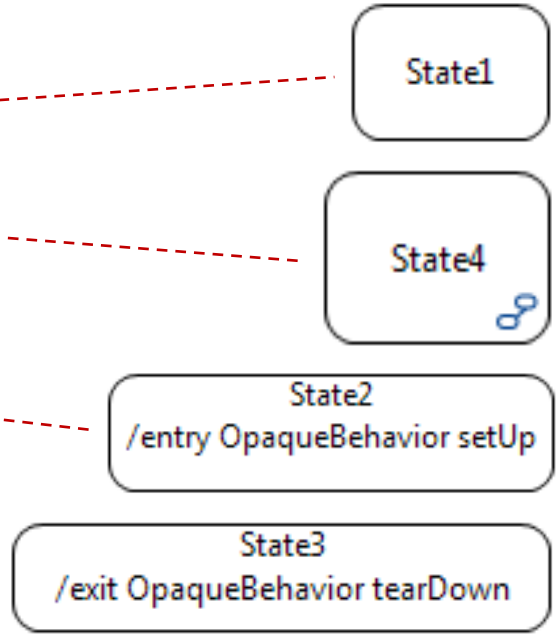
The original material for this section was developed by Prof. Juergen Dingel (Queen's University)

UML-RT/Papyrus-RT: Part III

- More on
  - State machines
    - States
      - Simple
      - Composite
    - Pseudo states
      - Initial
      - Choice point
      - Entry point
      - Exit point
      - History
      - Junction
  - Execution semantics
    - Run-to-completion
- Design guidelines

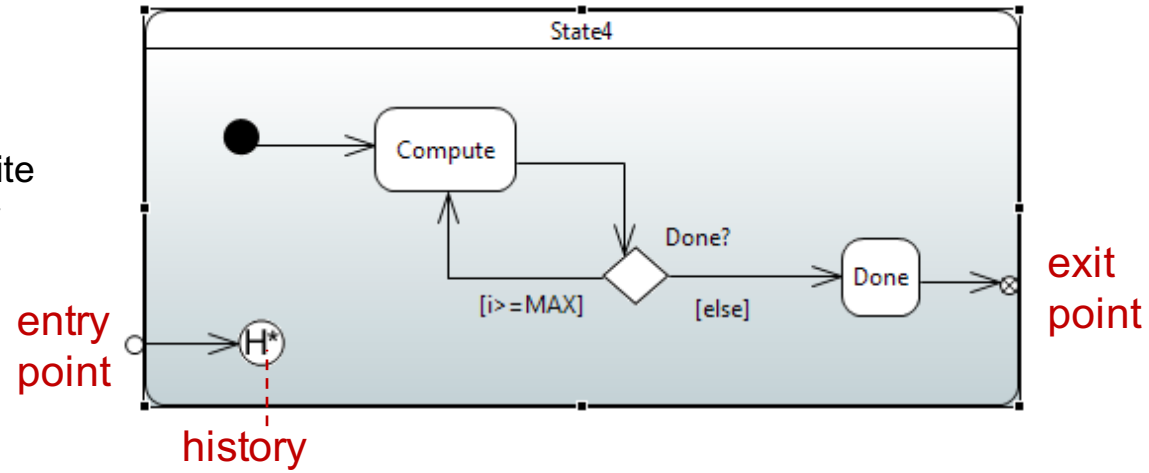
States II: Composite and Pseudo

- States
  - Kinds:
    - Simple
    - Composite (in hierarchical state machines)
  - May contain
    - Entry action
    - Exit action



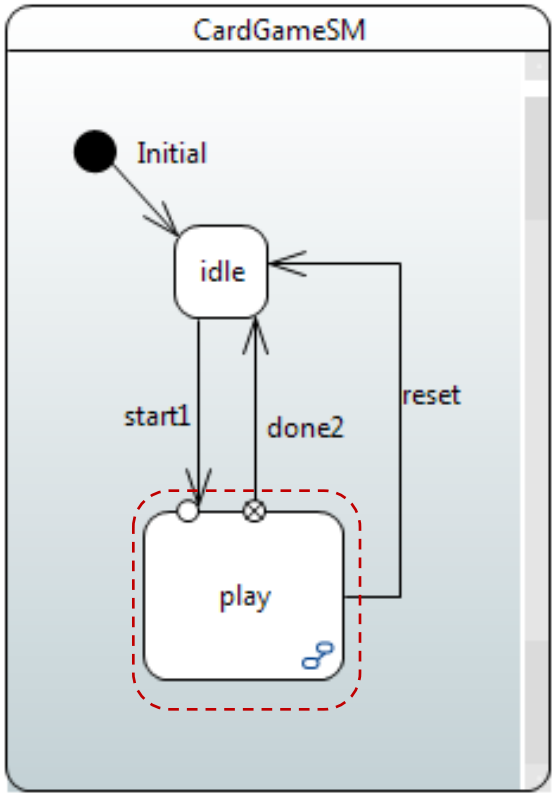
- Pseudo states
  - initial
  - choice point
  - history
  - entry point
  - exit point
  - junction point

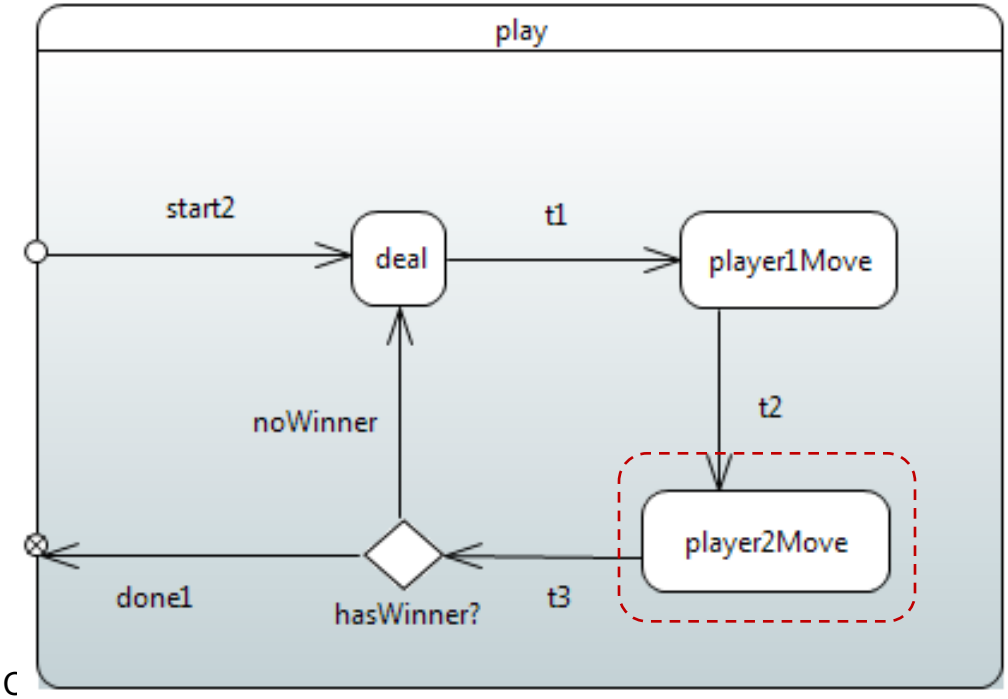
in composite states only



Group Transitions

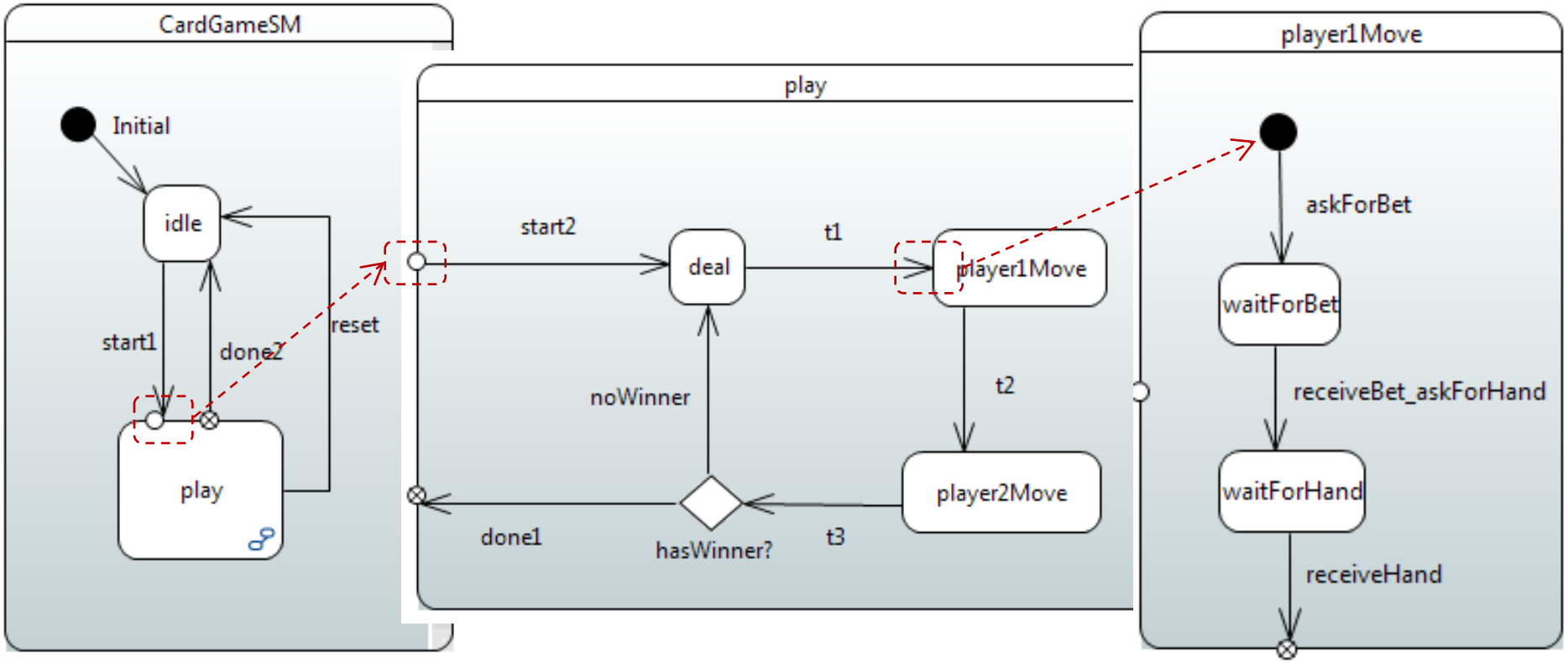
- Source state is composite
- Example:
  - Start configuration <'play','player2Move'>
  - Execute transition 'reset':
    - exit code 'player2Move', exit code 'play', effect 'reset', entry code 'idle'
  - End configuration <'idle'>





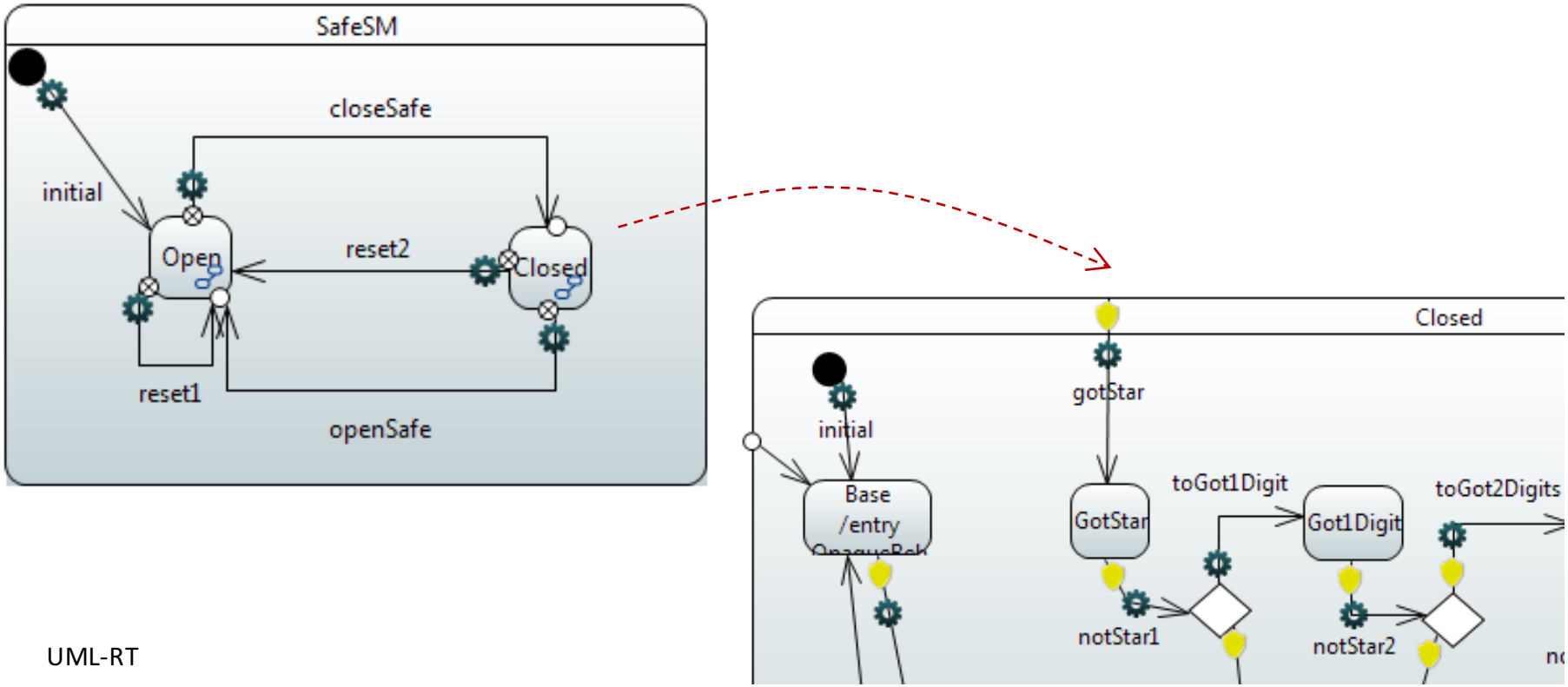
State Configuration

- States can be active: flow of control resides at state
- If a substate is active, its containing superstate is, too
- State configuration: list of active states
- Stable state configuration: no pseudo states and ends in basic state
- Example: <‘play’, ‘player1Move’, ‘waitForHand’>



Entry and Exit Points

- Required boundary pseudo states for transitions crossing boundaries of composite states
- Transition ending at entry point w/o outgoing transitions: implicit return to history



Run-to-Completion

- The event processing of state machines follows ‘run-to-completion’ semantics
- Dispatching of message triggers execution of possibly entire chain of transitions (‘`exec`’ on previous slide)
- Execution lasts until stable state configuration has been reached (last state in transition chain not a pseudo state)
- During transition execution, no other message will be dispatched

) execution triggered by message treated as one unit

) no ‘interleaved’ processing of messages

) less potential for bugs

Execution Semantics II

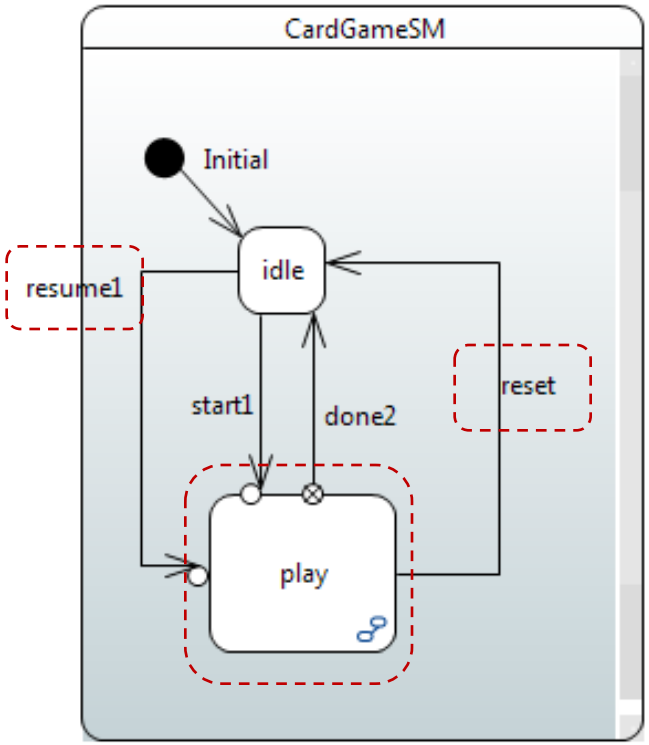
Controller main loop

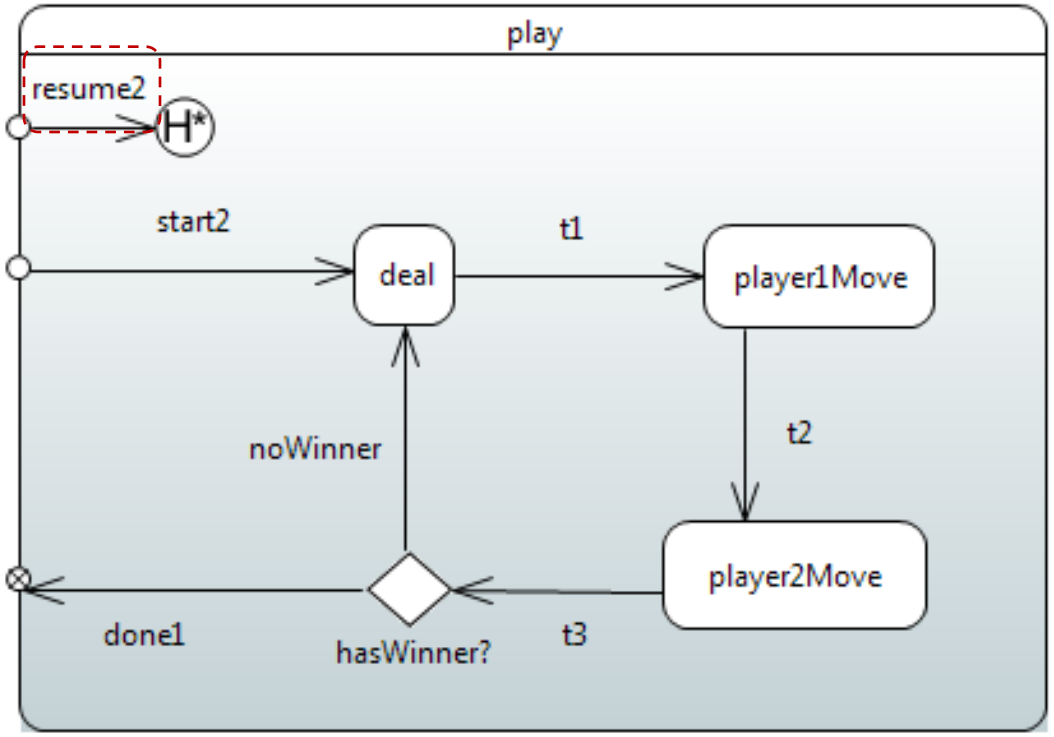
```
WHILE (1) {
  m = dequeue(MQ);
  IF can find transition t such that enabled(ssc,m,t) THEN ssc = exec(ssc,t);
  ELSE report 'Unexpected message m';
}
WHERE
enabled(ssc,m,t) = (1) source(t) is active, (2) trigger(t) matches m,
                   (3) eval(guard(t))='true', and
                   (4) source(t) does not contain any other state satisfying (1),(2),(3)
exec(ssc,t) = LET ssc=<s1, …, si-1, si, si+1, …, sn> where si=source(t) IN
                FOR j=n to i+1 {execute exit of sj}
                targetOfChain = execChain(t);
                sk = leastCommonAncestor(source(t), targetOfChain);
                LET <sk, s'1, …, s'm> be containment hierarchy where s'm=targetOfChain IN
                   RETURN <s1, …, sk-1, sk, s'1, …, s'm>
execChain(t) = execute exit of source(t), if any;
               execute effect of t, if any;
               execute entry of target(t), if any;
          WHILE target(t) is pseudo state {
             find t' such that source(t')=target(t) and eval(guard(t'))='true';
             execute exit of state(source(t')), if any;
                  execute effect of t', if any;
             execute entry of state(target(t')), if any;
                  t = t';
               }
               RETURN target(t);
```

UML2.5 Spec, Section 14.2.3
http://www.omg.org/spec/UML/2.5/PDF

History

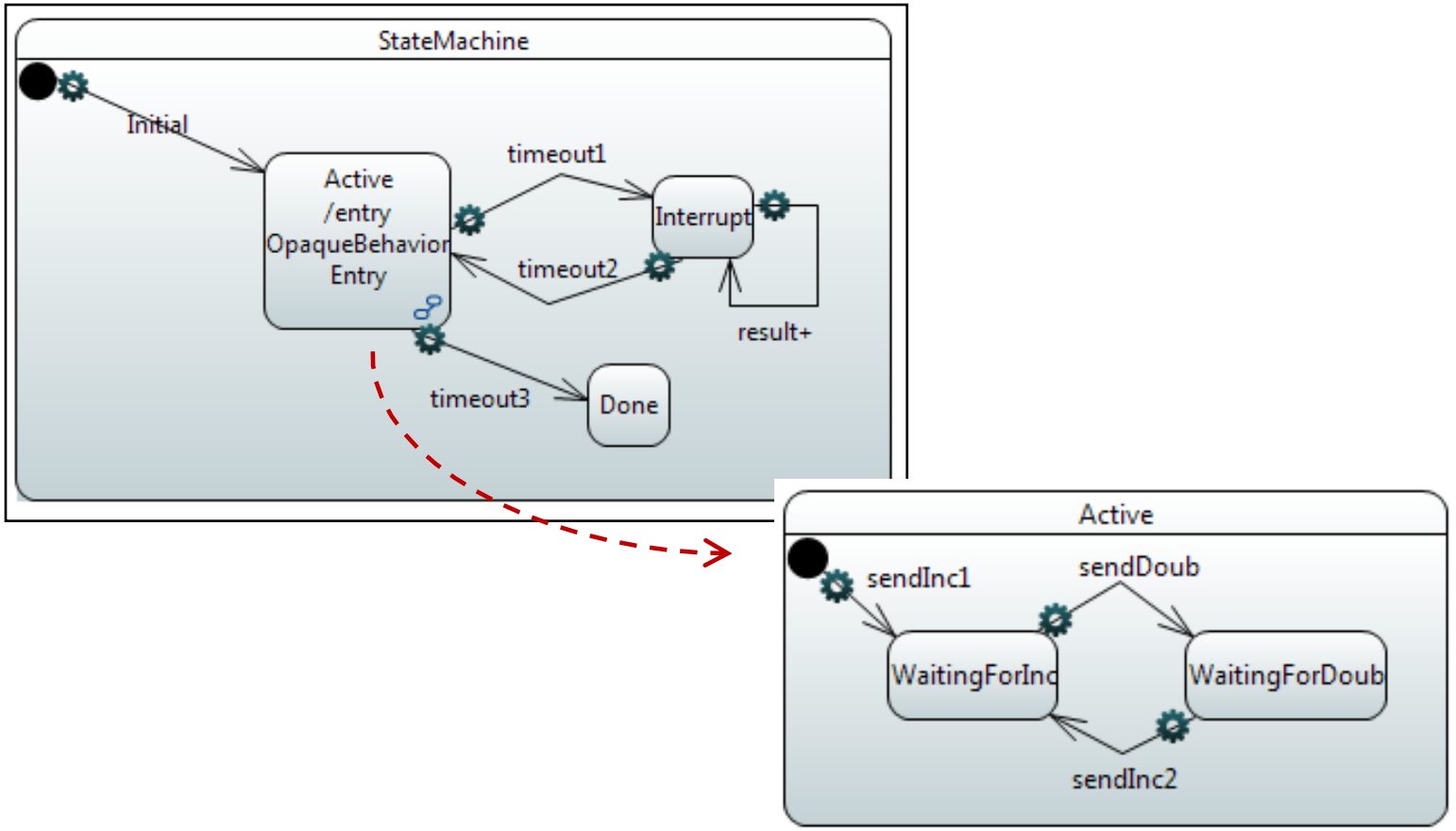
- Re-establish full state configuration that was active when containing state was active most recently
- If entering state for first time, go to initial state
- Example: from <'play', s> to <'play', s> with 'reset' 'resume1'





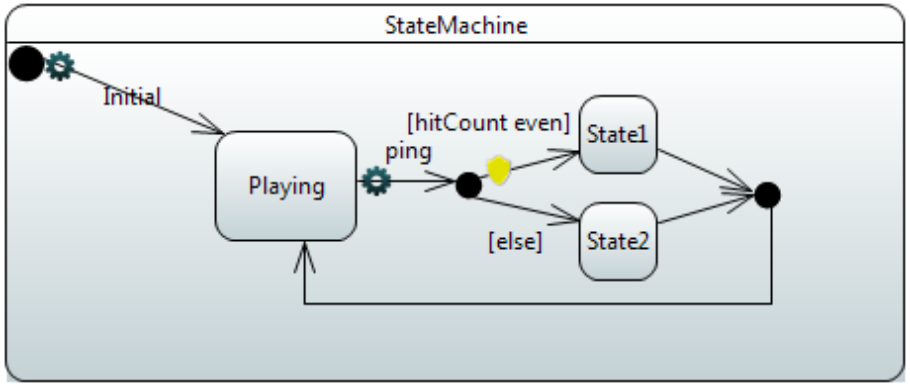
History (Cont'd)

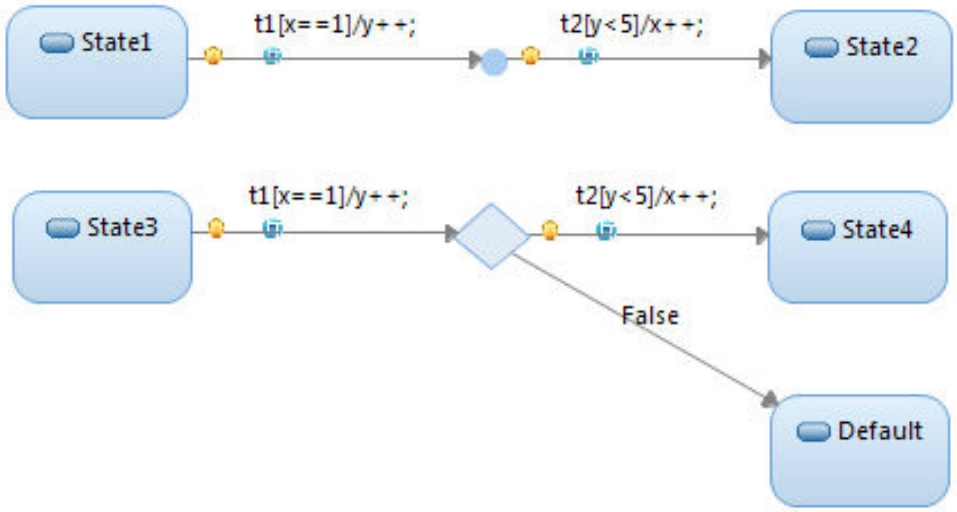
- History pseudo state does not need to be given explicitly
- Transition ends at boundary of composite state: Implicit return to history



Junction Points

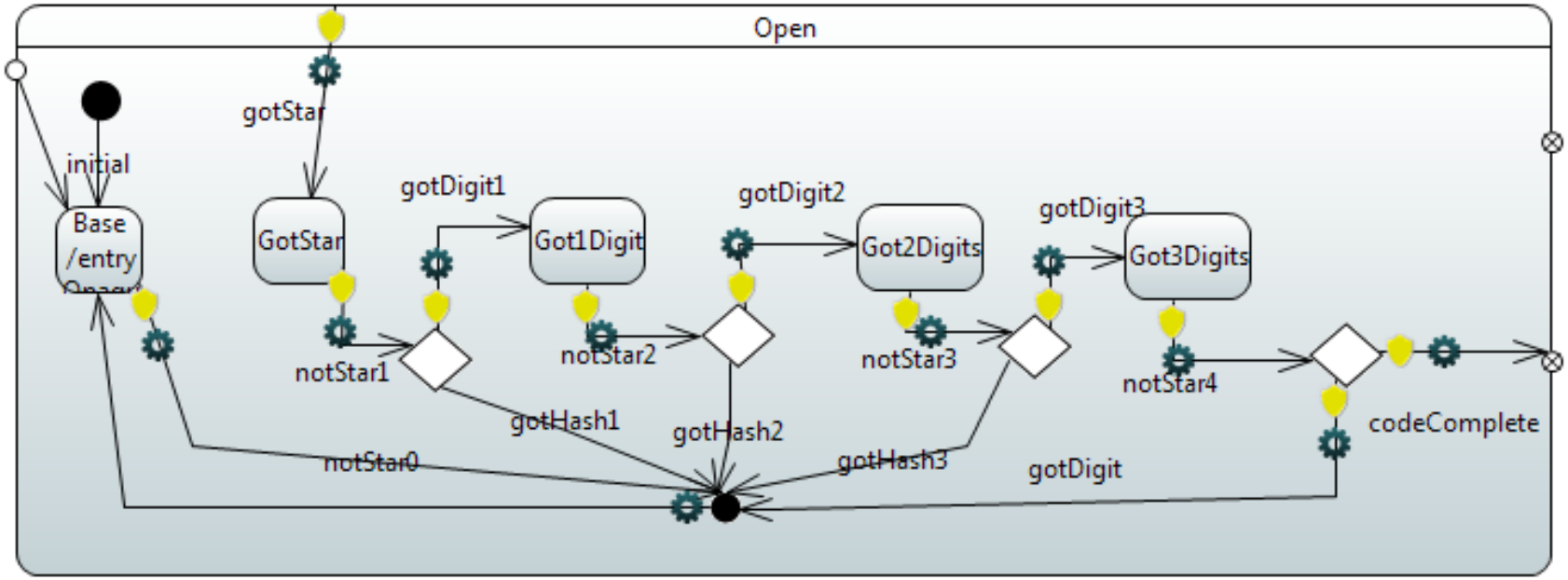
- Can be used to split and merge control flow
- Warning:
  - Static evaluation: All guards on transitions connected by junction points evaluated BEFORE first transition is taken
  - Transitions taken only when fully enabled path exists
- Choice points
  - Dynamic evaluation: Guards evaluated as transitions are executed
- Pros/cons?





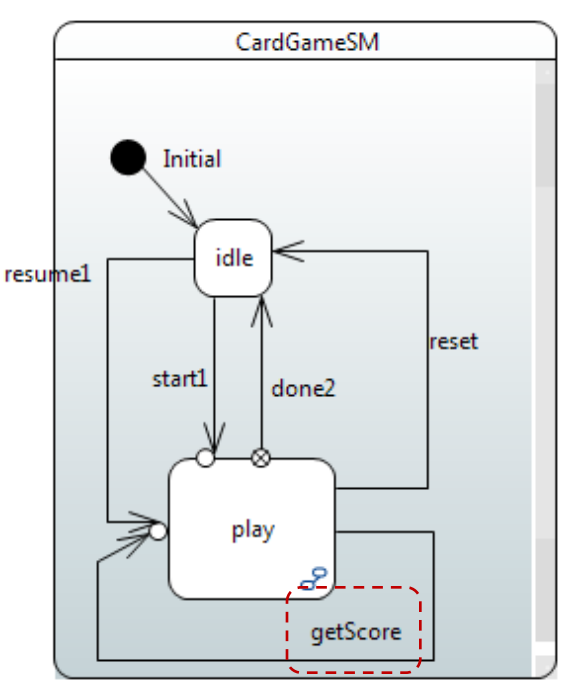
Junction Points (Cont'd)

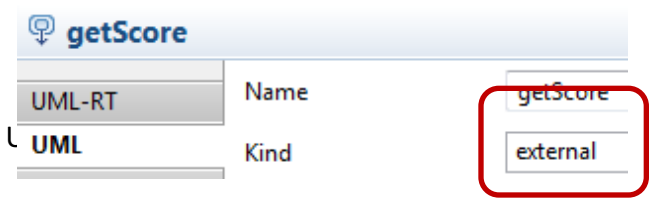
- Merge useful to avoid duplication of action

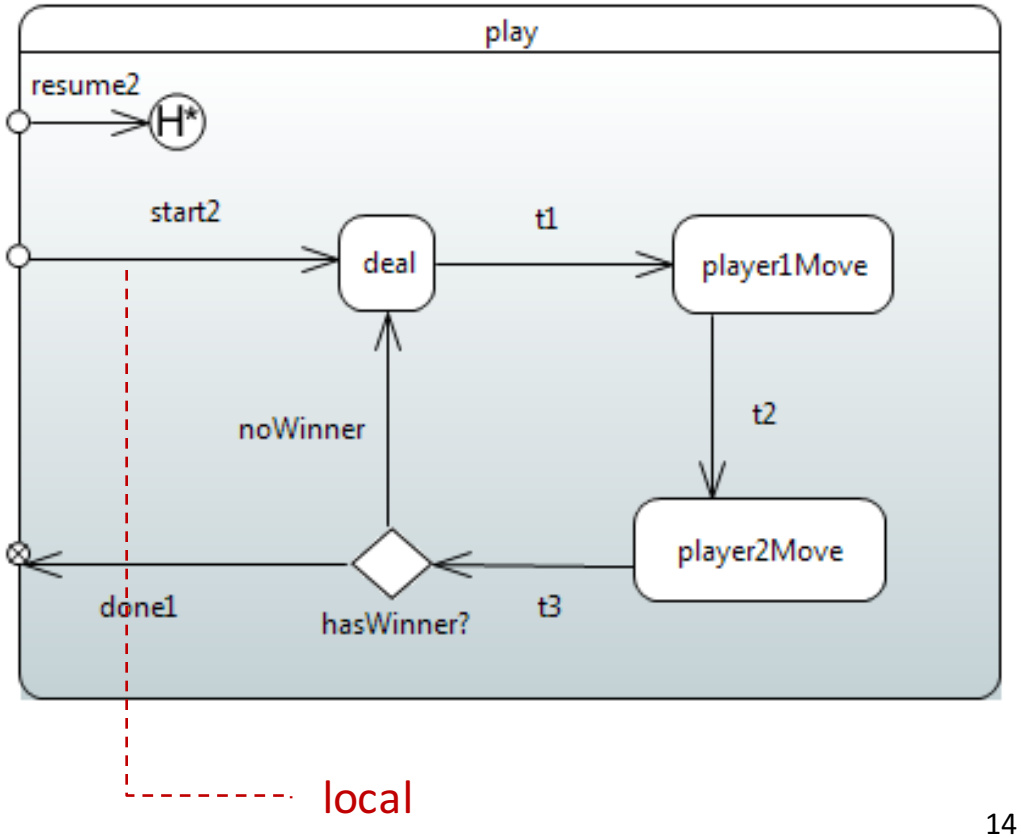


Transition Kinds

- 3 kinds: external, local, internal (relative to source state)
- External: source state (and all substates) exited and target state entered
- External self transition: external and source=target
- Local: source state contains transition, is not exited and source != target

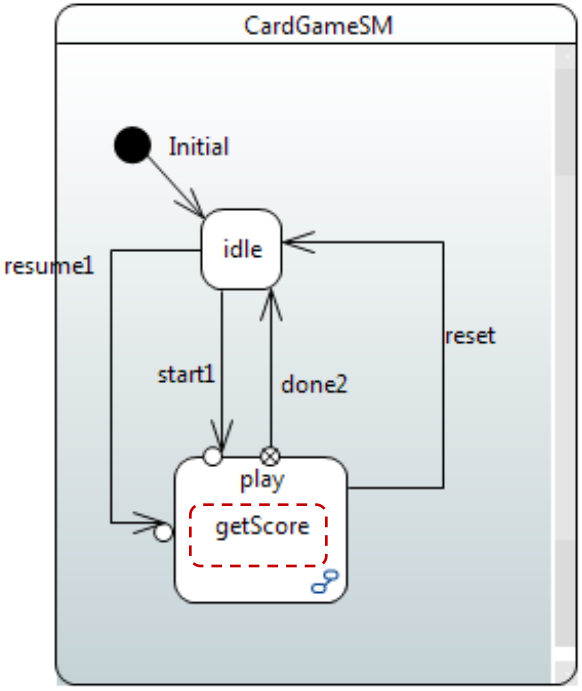


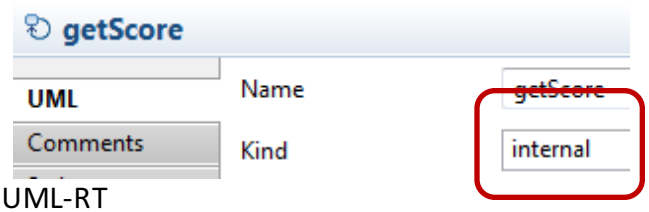


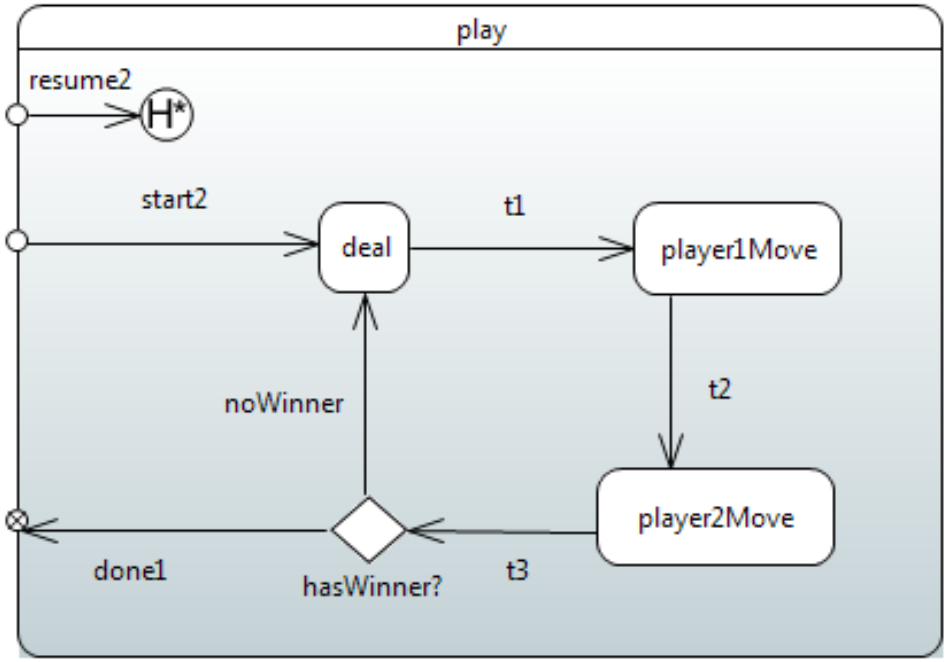


Transition Kinds (Cont'd)

- Internal:
  - Local transition with source==target
  - Source state (and all substates) remain active; no exit or entry actions executed







UML-RT: Design Guidelines

- General
  - Descriptive names
  - Readable, clear layout of models
  - Remove/cancel what is not needed anymore (timers, capsule parts)
  - Avoid duplication (through, e.g., operations, junction points for merging, entry and exit code)
- Capsules
  - Low coupling, high cohesion (look at connectors, message traffic, protocols)
  - Avoid overly deeply nested capsule definitions
- State machines
  - Avoid unreachable states and transitions
  - Avoid overly deeply nested composite states
  - Avoid composite states with only one substate

UML-RT: Design Guidelines (Cont'd)

- Action code
  - Short, simple, terminating, readable, avoid 'hidden' states (e.g., flags and complex control flow)
- Junction points
  - Only use for merging
- Transitions
  - Guards: short, simple, readable, side-effect-free
  - Out of choice points: guards exhaustive and exclusive
  - Out of initial, entry, exit, junction: no guard
  - Out of non-pseudo state: no guards
  - Use different kinds (external, local, internal) appropriately
  - Avoid dropped, 'unexpected' messages
  - Make copy of complex message parameters upon receipt
  - Can't cross 'state boundaries' w/o going through an entry or exit point

UML-RT: Design Guidelines (Cont'd)

- Observability
  - Insert informative log statements at suitable places to facilitate reasoning about the model (debugging, error localization)

    Format:

    ```
    Logger.log("[Name of capsule part](Name of state)…(Name of substate) info")
    ```

    where 'info' describes
    - message and/or data received, or
    - attribute values
  - Consider use of command-line parameters to facilitate testing