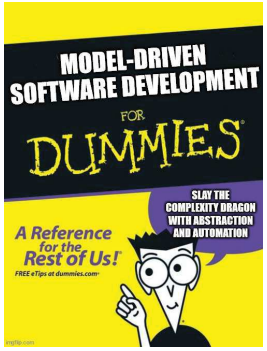
# CISC 844: Models in Software Development: Methods, Techniques and Tools




## UML-RT and Model RealTime: Part IV

**Juergen Dingel**
**Winter 2025**



## UML-RT/Model RealTime: Part IV

1. Transition kinds
   - External, local, internal
2. RTS
3. Dynamic change I
   - Capsules: 3 kinds of capsule parts
     - Fixed, optional, plugin
4. Multiplicity/replication
5. Defer/recall
6. Dynamic change II
   - Connectors: dynamic wiring w/ SAP/SPP

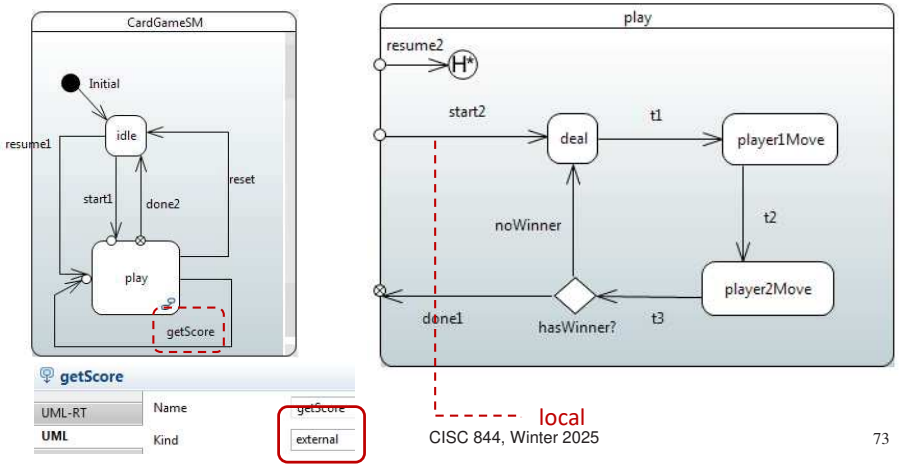
RTS library services

- System
  - `RTTiming.h`
  - `RTLog.h`
  - `RTFrame.h`
- Capsule
  - `RTActor.h`
- Communication
  - `RTOutSignal.h`
  - `RTMessage.h`
  - `RTProtocol.h`
  - `RTInSignal.h`



## Transition Kinds

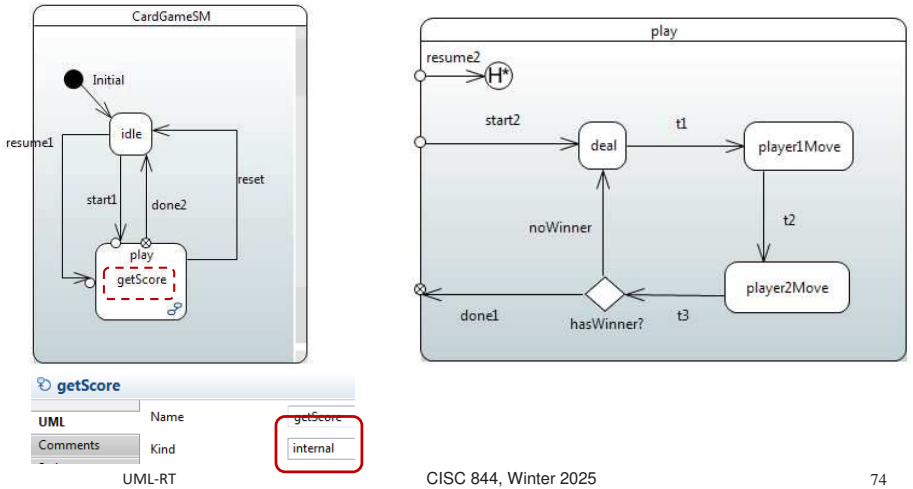
- 3 kinds: external, local, internal (relative to source state)
- External: source state (and all substates) exited and target state entered
- External self transition: external and source=target
- Local: source state contains transition, is not exited and source != target



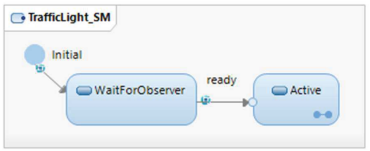


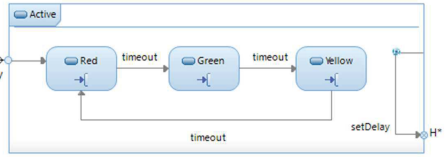

## Transition Kinds (Cont'd)

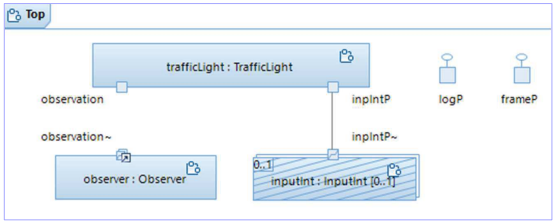
- Internal:
  - Local transition with source==target
  - Source state (and all substates) remain active; no exit or entry actions

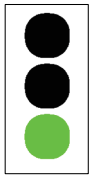


# Example: Internal Transitions



```
[inpInt] starting up, thread id: 570416
[inpInt] waiting for requests for input 'getInput'
[inpInt] please input an integer: 1
[inpInt] got 1
[inpInt] please input an integer: 2
[inpInt] got 2
[inpInt] please input an integer: 1
[inpInt] got 1
[inpInt] please input an integer:
```

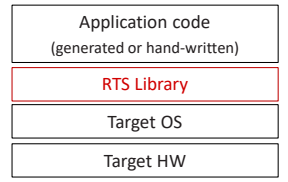


# Run Time Services (RTS) Library

- Provides services to application that involve resources managed by the RTS
  - Capsules, communication, timing, logging, frame
- Can be found in

  **<RSARTE Installation Directory>/eclipse/rsa_rt/C++/TargetRTS/src**

```
>> C:\Users\dingel\Programs\eclipse\rsa_rt\C++\TargetRTS\src> ls
include                 RTDiagStream            RTMemoryUtil            RTSymmetricSignal
MAIN                    RTDictionary            RTMessage               RTTcpInBuffer
RTAbortController       RTDynamicStringOutBuffer RTMessageQ             RTTcpOutBuffer
RTActiveConnector       RTElasticArray          RTObject_class          RTTcpSocket
RTActor                 RTEncoding              RTObserver              RTThread
RTActorClass            RTEnumerated            RTOBuffer               RTTime
RTActorId               RTEventInfo             RTOutSignal             RTTimerActor
RTActorProbe            RTException             RTPeerController        RTTimerController
RTActorRef              RTExceptionSignal       RTPointer               RTTimerId
RTActorRefProbe         RTExternal              RTPrefix                RTTimerList
RTActor_class           RTFieldDescriptor       RTPriority              RTTimerNode
RTArray_class           RTFloat                 RTProbe                 RTTimespec
RTAsciiDecoding         RTFormat                RTProtocol              RTTiming
RTAsciiEncoding         RTFrame                 RTProtocolAdapter       RTToolsetObserver
RTBoolean               RTGlobalSignal          RTProtocolDescriptor    RTTruncatingBuffer
RTByteBlock             RTIBuffer               RTPurgeFilter           RTTypedValue
RTCachedString          RTinet                  RTQFilter               RTUnknownObject
RTCharacter             RTInSignal              RTQueue                 RTVAsciiDecoding
RTCmdLineObserver       RTInteger               RTReal                  RTVAsciiEncoding
RTConnector             RTInterval              RTRecallFilter          RTWebObserver
RTController            RTIOMonitor             RTRelayPort             RTWrapper
RTCounts                RTJob                   RTResourceMgr           target
RTCustomController      RTJsonEncoding          RTRootProtocol          build.bat
RTDaemon                RTLayerConnector        RTSample                Build.pl
RTDaemonInfo            RTLayerData             RTSequence              lintnt.BAT
RTDataObject            RTLocalConnector        RTSequenceOf            main.mk
RTDebugger              RTLog                   RTSignal                main.nmk
RTDebuggerInput         RTLogBuffer             RTSoleController        Makefile
RTDecoding              RTMain                  RTStreamBuffer          makent.BAT
RTDiag                  RTMemoryInBuffer        RTString                MANIFEST.cpp
RTDiagBuffer            RTMemoryOutBuffer       RTSuperActor            Rational.mk
```



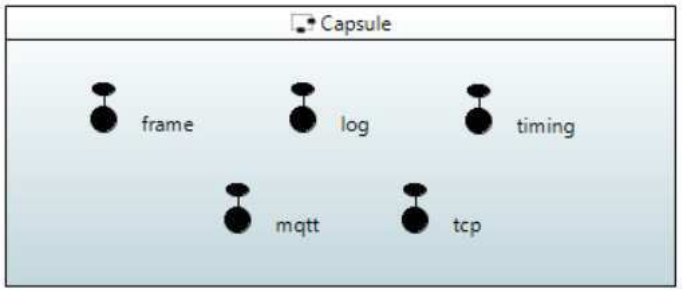



# The RTS is Actually Not That Large

```
PS C:\Users\dingel\Programs\eclipse\rsa_rt\C++\TargetRTS\src> more *\*.cc | wc -l
43509
PS C:\Users\dingel\Programs\eclipse\rsa_rt\C++\TargetRTS\src>
```



# Run Time Services (RTS) Library: System Ports



# Run Time Services (RTS) Library: Timer Services

- **`RTTiming`**
  - Type for timer ports
  - Methods
    - **`RTTimerNode informAt(RTTimespec)`**
      - 'One-shot' timer, absolute
      - Examples: `'RTTimespec now;`
        `RTTimespec::getclock(now);`
        `timer.informAt(now + RTTimespec(5, 0));'`
    - **`RTTimerNode informIn(RTTimespec)`**
      - 'One-shot' timer, relative
      - Example: `'timer.informIn(UMLRTTimespec(5, 0));'`
    - **`RTTimerNode informEvery(RTTimespec)`**
      - Periodic timer
      - Example: `'timer.informEvery(RTTimespec(5, 0));'`
    - **`cancelTimer(RTTimerId)`**
- **`RTTimespec`**
  - Supports comparison (e.g., '<','>=', '==') and simple manipulation (e.g., '+', '-')



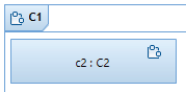
# Run Time Services (RTS) Library: Logging Services

- **`RTLog`**
  - Type of log ports
  - Methods
    - **`log(primitiveType)`**
      - With newline appended
    - **`show(primitiveType)`**
      - No newline appended
    - **`cr(int)`**
      - Output newlines

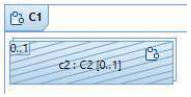


# Dynamic Change I: 3 Kinds of Capsule Parts

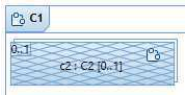
- 'Slot' = place in capsule where a part (i.e., capsule instance) can go
- Let c2:C2 be a part in capsule C1
- c2 fixed
  - Slot filled with instance of C2 when instance of C1 is created




- c2 optional
  - Slot can be filled/emptied by instance of C1 once at runtime
  - Once filled w/ instance of C2, instance will remain in same location for entire lifetime
  - Methods: **`incarnate`**, **`destroy`** (**`RTFrame.h`**)



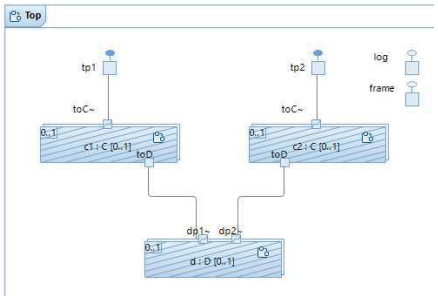

- c2 plugin
  - Slot that can be filled/emptied repeatedly at runtime with possibly different instances i2 of C2
  - i2 can also fill different slot at the same time, i.e., be shared
  - Methods: **`import`**, **`deport`** (**`RTFrame.h`**)





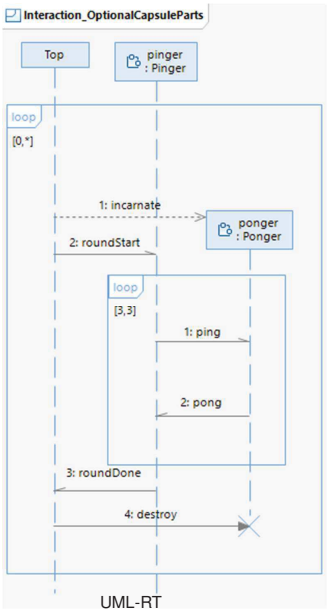

# Run Time Services (RTS) Library: Frame Services

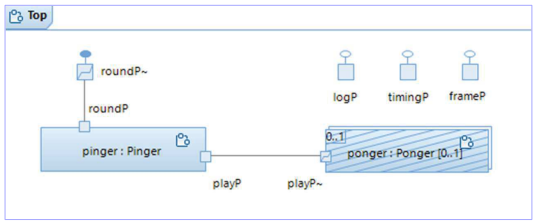
- **`RTFrame (RTFrame.h)`**
  - Type of frame ports
  - Methods: optional capsule parts
    - **`RTActorId incarnate(RTActorRef & cp)`**
      - **`cp`** is the capsule part into which to insert capsule instance
      - The capsule to incarnate is determined from the type of the part
    - **`bool destroy(RTActorId)`**
  - Example




```
log.log("[Top] starting up");
log.log("[Top] incarnating part 'c1'");
RTActorId id1 = frame.incarnate(c1);
log.log("[Top] incarnating part 'c2'");
RTActorId id2 = frame.incarnate(c2);
log.log("[Top] incarnating part 'd'");
RTActorId id3 = frame.incarnate(d);
log.log("[Top] sending 'go' to 'c1'");
tp1.go().send();
log.log("[Top] sending 'go' to 'c2'");
tp2.go().send();
```

# Example: 3xPingPong + Dynamic Instantiation

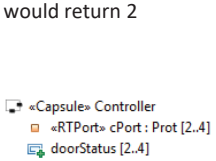


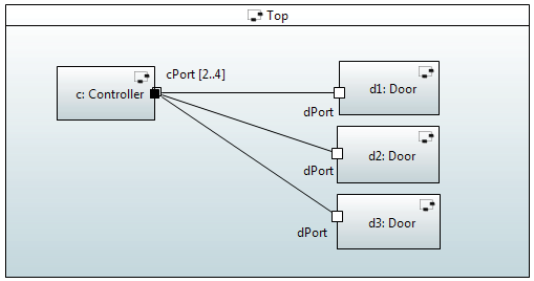






# Multiplicity/Replication

- Some elements can be replicated by setting the multiplicity
  - Attributes, ports, and parts (all instances of UML-meta type 'Property')
- Port replication
  - To send `m` to all doors: `cPort.m().send()`
  - To send `m` to a single, specific door (e.g., `d3`): `cPort.m().sendAt(2)`
  - To tell which port a message came in on: `msg->sapIndex0()` returns port index
    - E.g., if `d3` sends `m` to `c`, then `msg->sapIndex0()` in effect of transition triggered by `m` would return 2

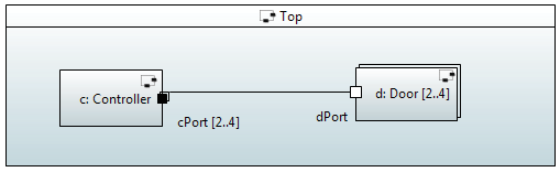


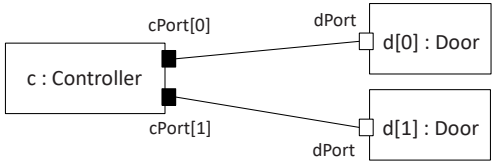



# Replication (Cont'd)

- Combining port and capsule replication
  - E.g.,



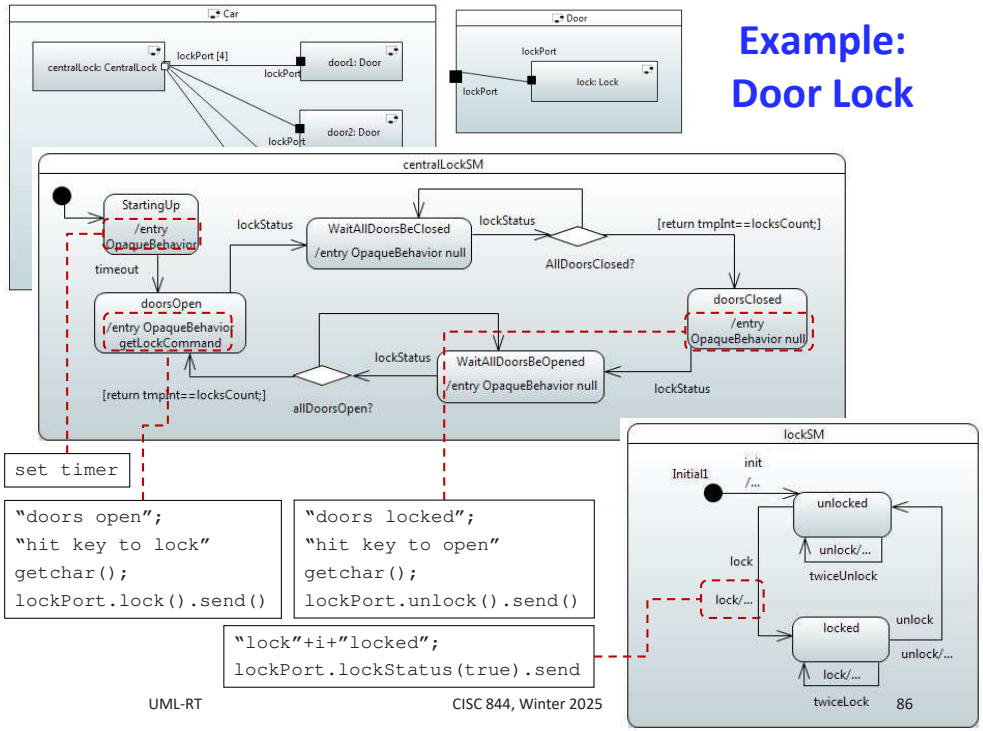

represents n **`Door`** instances each with a single port **`dPort`** that connects them to one of the n instances of **`cPort`**, where $2 \leq n \leq 4$. E.g.,




- Replication ranges [m..n] with m<n particularly useful when using dynamic capsule creation (optional and plugin capsules)

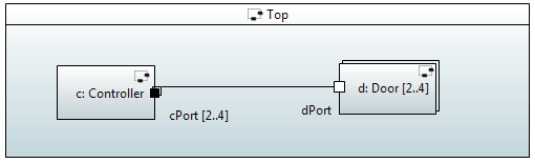


# Example: Door Lock



# Run Time Services (RTS) Library: Capsules

- **`RTActor`** (in **`rtactor.hh`**)
  - Methods
    - **`string getTypeName()`**
      - name of capsule
    - **`string getName()`**
      - name of capsule part
    - **`int getIndex()`**
      - index of capsule part; interesting when capsule is replicated
  - Example
    - In Controller: `logger.log("[%s:Controller] Starting up", this->getName());`
    - In Door: `logger.log("[d%s: Door] Starting up", this->getIndex());`






# Run Time Services (RTS) Library: Communication (1)

- **`RTOutSignal`**
  - Methods
    - **`bool send(priority)`**
      - asynchronous
      - priority argument optional
      - if port replicated, send over all instances
    - **`bool sendAt(index, priority)`**
      - to specific instance of replicated port (indices are 0-based)
    - **`int invoke(replyMsg)`**
      - synchronous, i.e., sender blocks until reply is received (via **`reply()`**)
      - mimicks 'operation call'
  - Properties
    - Messages sent over same connector received in same order they've been sent
    - Delivery of messages to unbound ports will fail
    - Delivery of messages that don't trigger transition, will be dropped with warning
    - If message data (accessible in entire transition chain via **`*rtdata`**) has type descriptor, it will be copied and passed by value
    - Message can have at most one parameter (wrap multiple values into data class)



# Run Time Services (RTS) Library: Communication (2)

- **`RTMessage`**
  - Base type for messages
    - Created upon send signal event; refers to signal being sent and its 'payload'
    - Signals separated from messages, so that different messages can refer to same signal (for broadcast signals)
  - Methods
    - **`bool defer()`**
      - Put message into 'defer queue'
- Aside: 'signals' vs 'messages'
  - Signals: elements defined in the protocol
  - Message:
    - represents the sending of a signal
    - contains a signal and any 'payload'

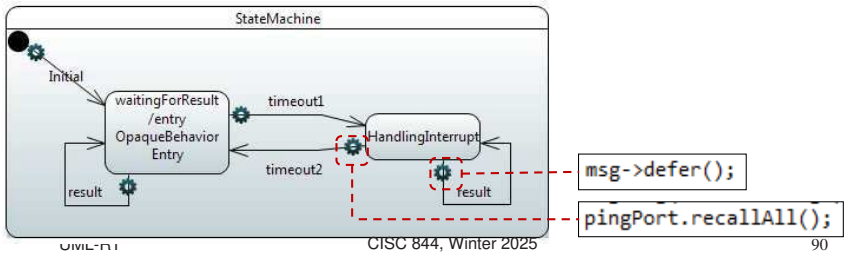
    ⇒ different messages can refer to same signal



# Defer/recall

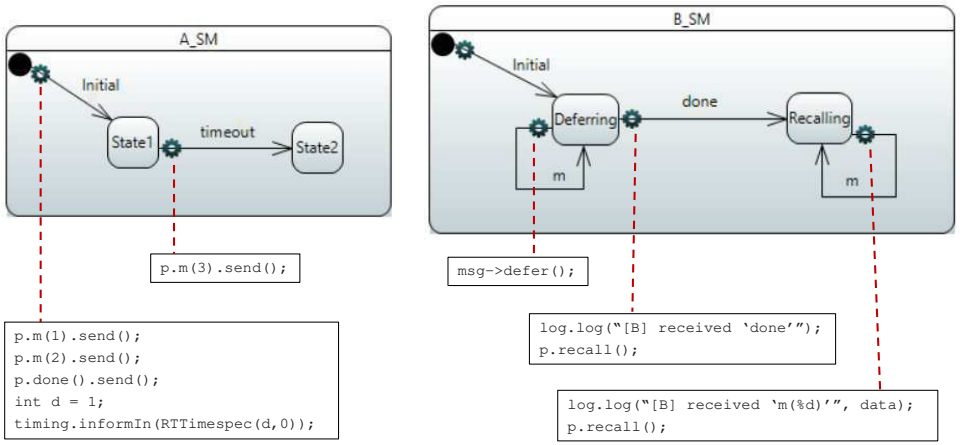
- Allows handling of messages that arrive while in 'wrong' state
- Defer m message on port p:
  - 'Wrong' state has self transition triggered by m with effect **`'msg->defer()'`**
- Recall message m on port p:
  - When entering state in which m should be handled, execute **`'p.recall()'`**, **`'p.recallAll()'`**, or **`'p.recallFront()'`**

| Port | Protocol Message |
|---|---|
| pingPort | in result |

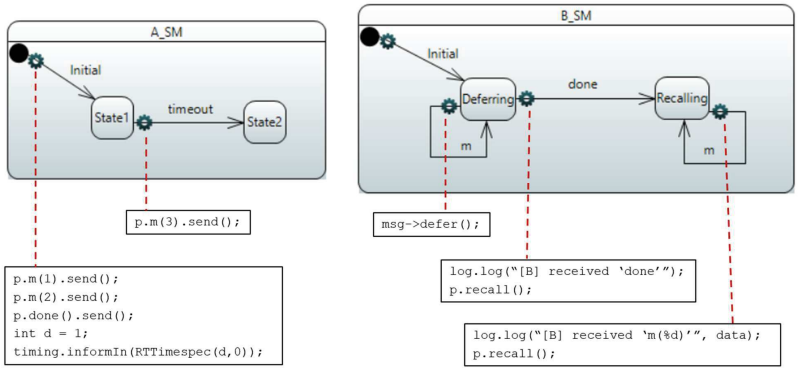


# Defer/recall (Cont'd)

- Note: Deferred message m will be 'overtaken' by messages arriving while m is in defer queue (can use `'recallFront()'`)



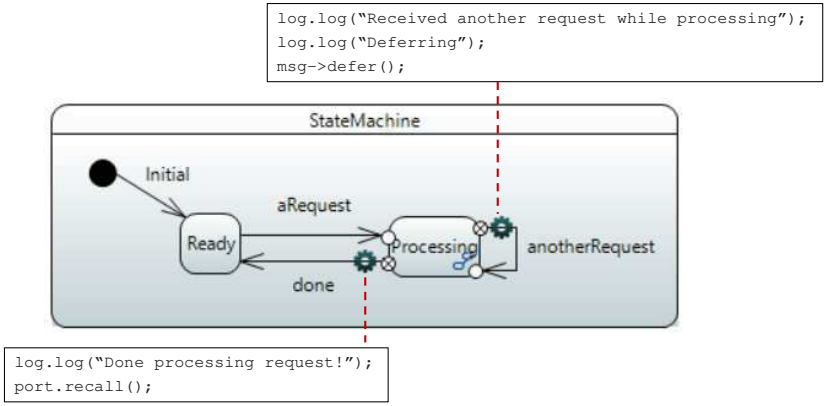





# Ways to Avoid 'Dropped Messages'

- Internal transition with trigger set to 'any event' (i.e., '*'), or
- Use 'defer/recall'
  - In effect code of self transition of 'Processing': **`msg->defer();`**
  - Then, when done with processing: **`port.recall();`**






# Run Time Services (RTS) Library: Communication (3)

- **`RTProtocol`**
  - Base type for protocols [RSARTE C++ Services Library, pages 53-57]
  - Methods
    - **`int recall(void)`**
      - Recall a deferred message on all instances of this port. Returns number of messages recalled (0 or 1). Calling recall on a port gets the first deferred message from one of the port instances. Messages are recalled behind other queued messages.
      - This function recalls the first deferred message on any port instance. To recall the first message on one specific port instance of a replicated port, use **`RTProtocol::recallAt()`**.
      - E.g., **`port1.recall()`** recalls first deferred message on any instance of the port named port1
    - **`int recallFront()`**
      - Recalls the first deferred message on any port instance. Calling **`recallFront`** on a port gets the first deferred message from one of the port instances, starting from the first (instance 0). Messages are recalled to the front main queue. Returns the number of recalled messages (0 or 1).
    - **`int recallAll(void)`**
      - Recall all deferred messages for the event on all port instances. Returns the number of recalled messages. To recall all messages on only one instance of a port with replicationfactor > 1, use **`RTProtocol::recallAllAt()`**
    - Also, **`recallAt()`**, **`recallAllAt()`**, **`recallAllFront()`**, **`purge()`**, and **`purgeAll()`**

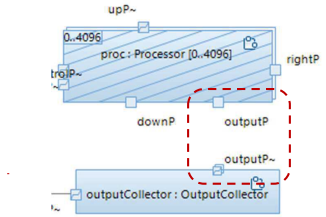
# Run Time Services (RTS) Library: Communication (4)

- **`RTInSignal`**
  - Base type for incoming signals [RSARTE C++ Services Library, pages 44-46]
  - Methods
    - **`int recall(int front=0);`**
      - E.g., **`p.m().recall(1)`**
      - Recall one deferred message for the event on all port instances.
      - front [optional] is a boolean int that indicates whether the message should be recalled to the front of the system message queue. If false, or left unspecified, the message is sent to the back of the message queue. By recalling to the front, it is possible to avoid overtaking of messages.
      - Returns the number of recalled messages. There is no time-limit on deferral. Applications must take precautions against forgetting messages on defer queues.
      - This function recalls the first deferred message of this event on any port instance.
    - **`int recallAll(int front=0);`**
      - Recall all deferred messages for the event on all port instances. Returns the number of recalled messages. This function recalls ALL deferred messages of this event on ALL port instances
    - **`int recallAllAt(int index, int front=0);`**
      - Recall all deferred messages on a specific port instance
    - to delete deferred messages, **`purge()`** and **`purgeAt(int index)`** can be used



# Dynamic Change II: Unwired Ports

- So far, only wired ports
  - Connected automatically when instances are created
- Unwired ports
  - Connected at run-time via 'publish/subscribe'
    - Port on publisher: Service Provision Point (SPP), registration kind = application
    - Port on subscriber: Service Access Point (SAP), registration kind = application
    - Register with RTS using unique service name (manually or automatic)




```
10 // connecting with outputCollector
11 outputP.registerSAP("OutputCollector_Processor");
```

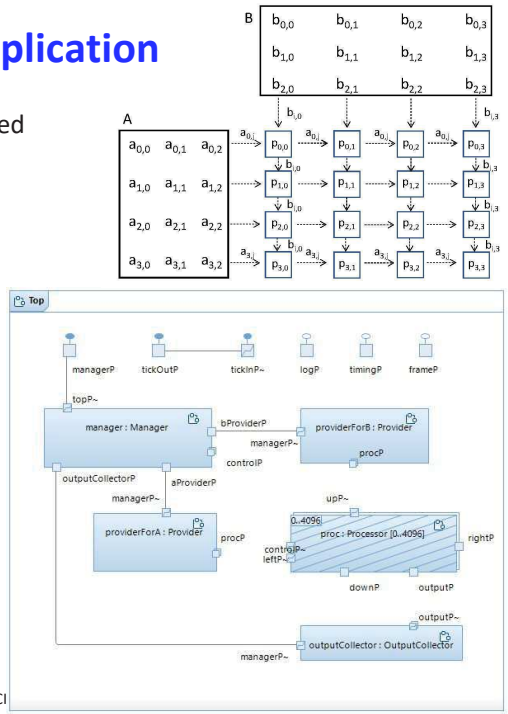
```
87 // connect with processors
88 outputP.registerSPP("OutputCollector_Processor");
```



# Example: Matrix Multiplication

- UML-RT features/constructs used
  - optional capsules
  - uynamic wiring

  but also
  - choice points
  - command line arguments
  - data classes for message arguments
  - operations
  - defer/recall
  - replication
  - tick pattern
- Available on the sample models page

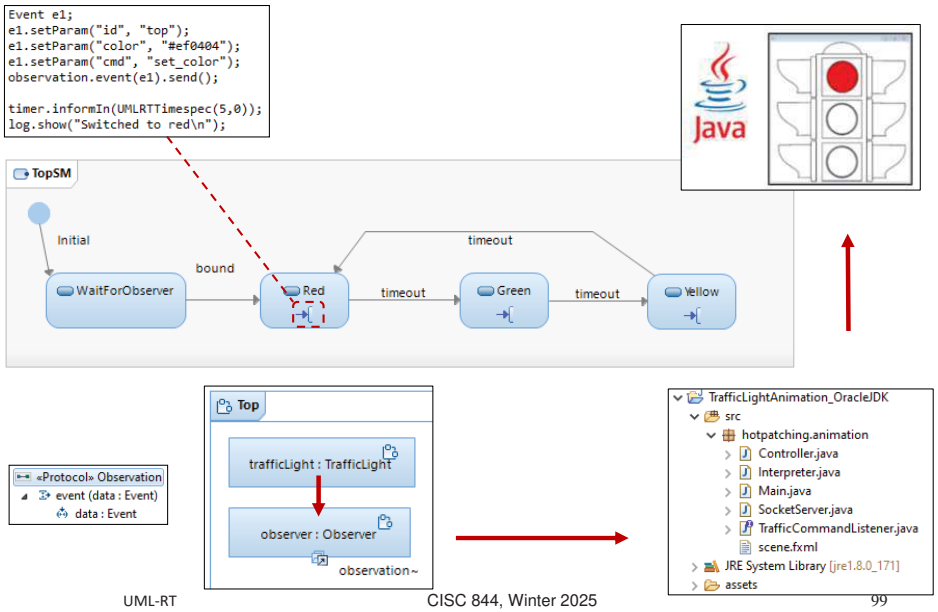





# Run Time Services (RTS) Library: Communication (5)

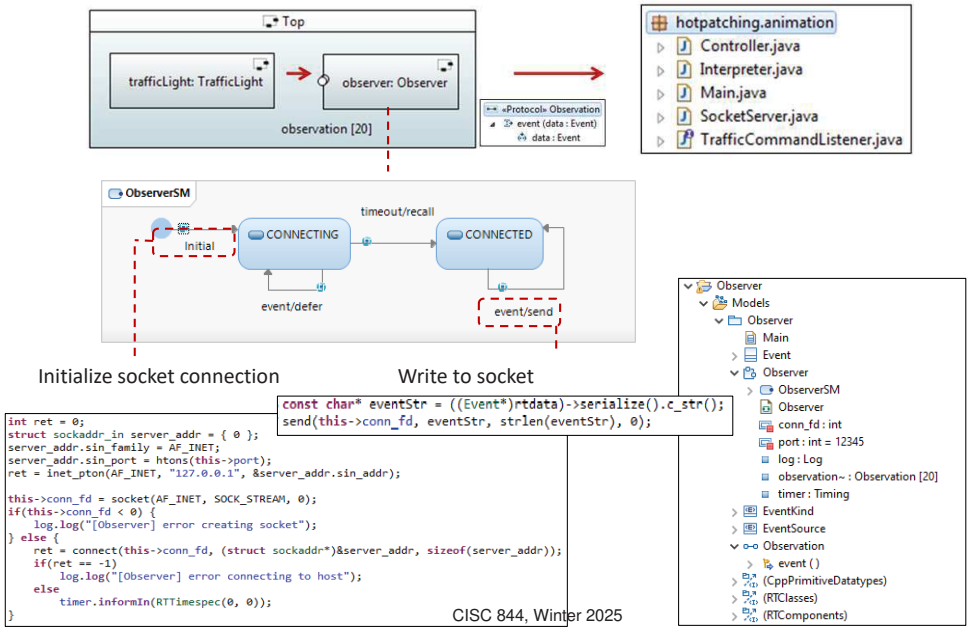
- **`RTProtocol`**
  - Base type for protocols
  - Methods
    - **`bool recall()`**
    - **`bool recallAll()`**
    - **`bool recallFront()`**
    - **`bool registerSAP(string)`**
      - Non-wired ports with 'RegistrationKind=Application' have to be wired programmatically
      - Registers this port as SAP port with RTS to allow for dynamic binding from SPP
      - Example: **`p1.registerSAP("myService");`**
    - **`bool registerSPP(string)`**
      - Registers port as SPP providing service with name 'string' and automatically connects with matching SAP ports (e.g., **`p2.registerSPP("myService");`** )
      - Typically, one SPP port and multiple SAP ports
    - **`bool deregisterSAP(), bool deregisterSPP()`**

# How Does the Observer Work?





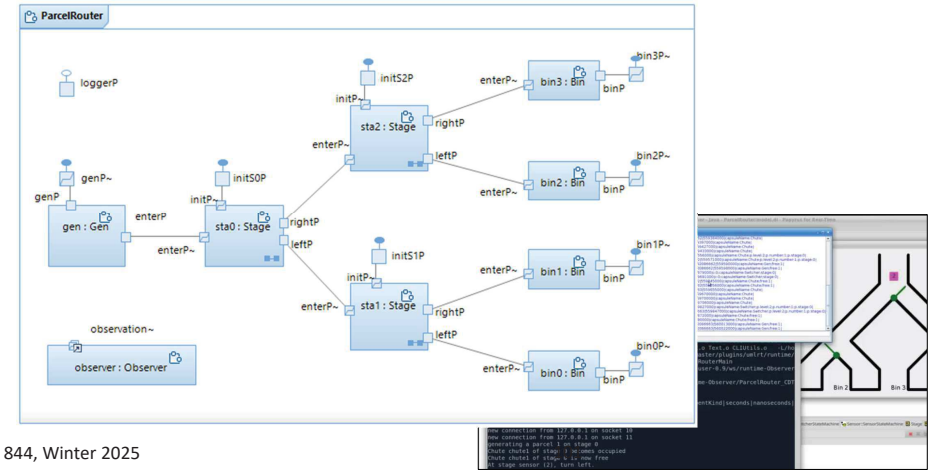

# How Does the Observer Work? (Cont'd)






# Observer Capsule: Examples

- Monitoring and steering
  - Parcel routing system
  - https://www.youtube.com/watch?v=EbMIgEX9O58






# Run Time Services (RTS) Library: Utilities

- **`RTMain`**
  - **`static int getArgCount();`**
    - Get number of user-supplied command line arguments
  - **`static const char * getArg(int index);`**
    - Get argument with particular index
  - Example:
    - **`const char *arg0 = RTMain::getArg(0);`**

# Project Suggestions I: Applications of Model RealTime

- Build sample application, e.g.,
  - (simulation of)
    - some reactive system (e.g., game, traffic/elevator control, vending machine, ATM, digital watch, garage door, microwave), or
    - standard concurrent/distributed systems example (e.g., consensus, sleeping barbers, distributed hash tables), or
    - discrete event control example (e.g., air traffic control, cat & mouse)
  - with or without, e.g.,
    - Animation (e.g., using SDL2)
    - advanced UML-RT features (plugin capsules, inheritance, multi-threading, priorities)
    - advanced M-RT features (e.g., tracing and debugging [Moh10f, HCL20a])
    - support for distributed development (via EGit)
    - external components (C++ libraries, analytics, learning)
  - to
    - illustrate or explore (e.g., basic or advanced UML-RT or M-RT features, use of external components, fault-tolerance or adaptation technique)



# Project Suggestions II: Extensions of UML-RT

- Explore HCL extensions and sample systems
  - Traffic light, https://github.com/HCL-TECH-SOFTWARE/qt-traffic-light
  - Distribution
    - https://github.com/HCL-TECH-SOFTWARE/lib-tcp-server
    - https://github.com/HCL-TECH-SOFTWARE/pingpong-distributed
  - Capsule unit testing (using Mocha)
    - https://model-realtime.hcldoc.com/help/topic/com.ibm.xtools.rsarte.webdoc/Articles/Testing/Capsule%20Unit%20Testing.html?cp=23_2_16_0
- Explore open-source, web-based, graphical/textual-hybrid version
  - HCL: https://www.hcl-software.com/devops-code-realtime



# Project Suggestions III: DSLs

- Extend Assignment 4
  - e.g., improve validation
- Use Xtext to develop or investigate
  - a new DSL
- Explore use of other language workbench and compare
  - textX (Python)
  - Langium (Java)
  - JetBrains MPS
  - Eclipse Theia
    - Platform to develop multi-language web-based IDEs
- Explore (extensions for) existing DSLs
  - data science (e.g., Knime)
  - machine learning
  - …



# Project Suggestions IV: Other

- Explore actor languages
  - Erlang, Elixir, Akka, Pykka, Orleans, …
- Design recovery
  - Open-source Space Invader implementations
- ?
  - Talk to me