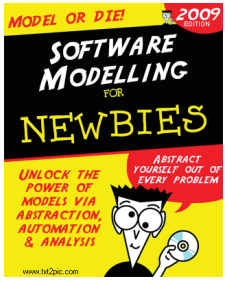
CISC836: Models in Software Development: Methods, Techniques and Tools



UML-RT and RSARTE: Part IV

Juergen Dingel
Feb 2020

UML-RT/RSARTE: Part IV

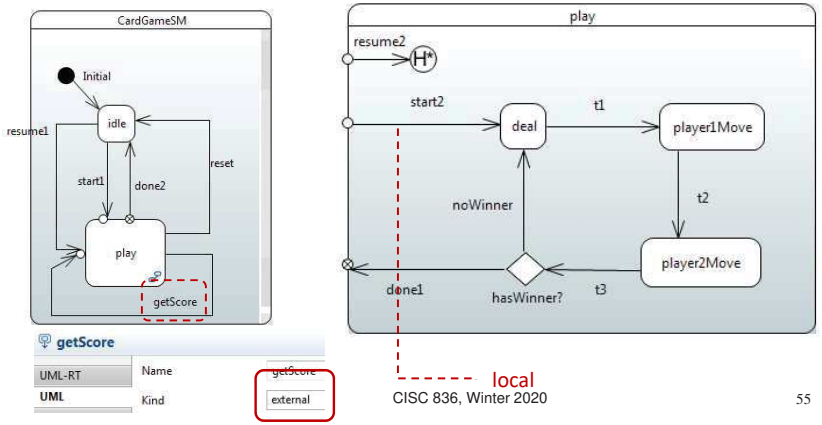
1. Transition kinds
   - External, local, internal
2. RTS
3. Dynamic change I
   - Capsules: 3 kinds of capsule parts
     - Fixed, optional, plugin
4. Multiplicity/replication
5. Defer/recall
6. Dynamic change II
   - Connectors: dynamic wiring w/ SAP/SPP

RTS library services

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

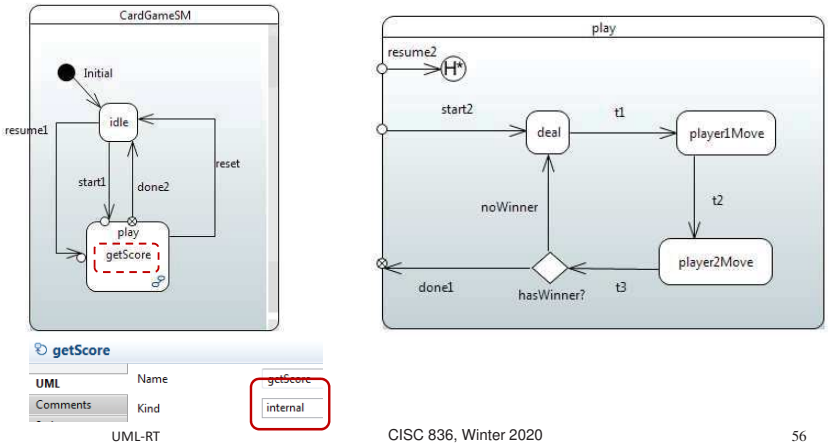
Transition Kinds

- 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

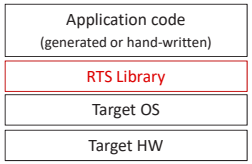


Run Time Services (RTS) Library

- 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

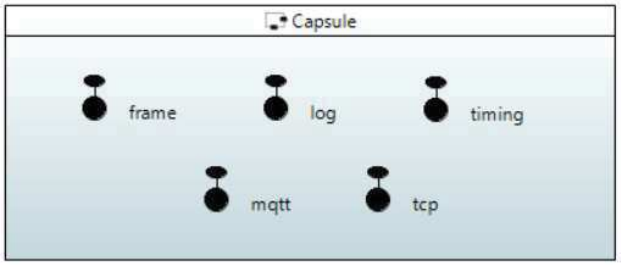
```
>> C:\Users\dingel\Programs\eclipse\rsa_rt\C++\TargetRTS\src> ls
include                 RTDiagStream              RTMemoryUtil            RTSymmetricSignal
MAIN                    RTDictionary              RTMessage               RTTcpInBuffer
RTAbortController       RTDynamicStringOutBuffer  RTMessageQ              RTTcpOutBuffer
RTActiveConnector       RTElasticArray            RTObject_class          RTTcpSocket
RTActor                 RTEncoding                RTObserver              RTThread
RTActorClass            RTEnumerated              RTOutBuffer             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

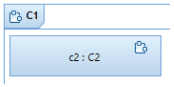
- **RTTiming**
  - Type for timer ports
  - Methods
    - **RTTimerNode informAt(RTTimespec)**
      - 'One-shot' timer, absolute
      - Examples: `'RTTimespec now;`
        `RTTimespec::getclock(now);`
        `timer.informAt(now + RTTimespec(5, 0));'`
    - **RTTimerNode informIn(RTTimespec)**
      - '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

- Part = 'slot'
- Let c2:C2 be a part in capsule C1
- c2 fixed
  - Slot filled with instance of C2 when instance of C1 is created

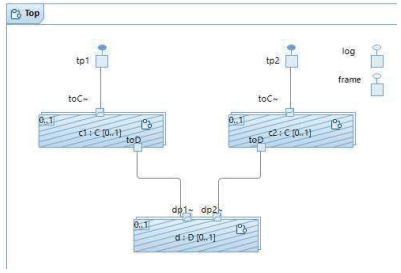


- c2 optional
  - Slot can be filled/emptied by instance of C1 at runtime
  - Once filled w/ instance of C2, instance will remain in same location for entire lifetime
  - Methods: **`incarnate`**, **`destroy`** (**`RTFrame.h`**)

- 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

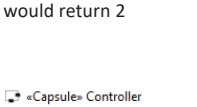
- **`RTFrame (RTFrame.h)`**
  - Type of frame ports
  - Methods: optional capsule parts
    - **`RTActorId incarnate(RTActorRef & cp)`**
      - **`cp`** is the capsule part into which to insert capsule instance
      - The capsule to incarnate is determined from the type of the part
    - **`bool destroy(RTActorId)`**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

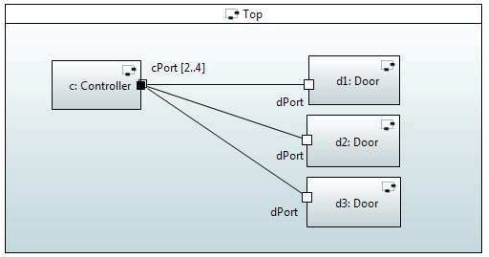



```
log.log("[Top] starting up");
log.log("[Top] incarnating part 'c1'");
RTActorId id1 = frame.incarnate(c1);
log.log("[Top] incarnating part 'c2'");
RTActorId id2 = frame.incarnate(c2);
log.log("[Top] incarnating part 'd'");
RTActorId id3 = frame.incarnate(d);
log.log("[Top] sending 'go' to 'c1'");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


Multiplicity/Replication

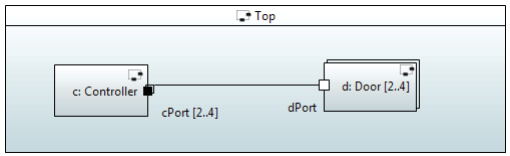
- Some elements can be replicated by setting the multiplicity
  - Attributes, ports, and parts (all instances of UML-meta type 'Property')
- Port replication
  - To send **`m`** to all doors: **`cPort.m().send()`**
  - To send **`m`** to a single, specific door (e.g., **`d3`**): **`cPort.m().sendAt(2)`**
  - 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



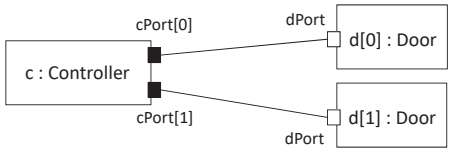


Replication (Cont'd)

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

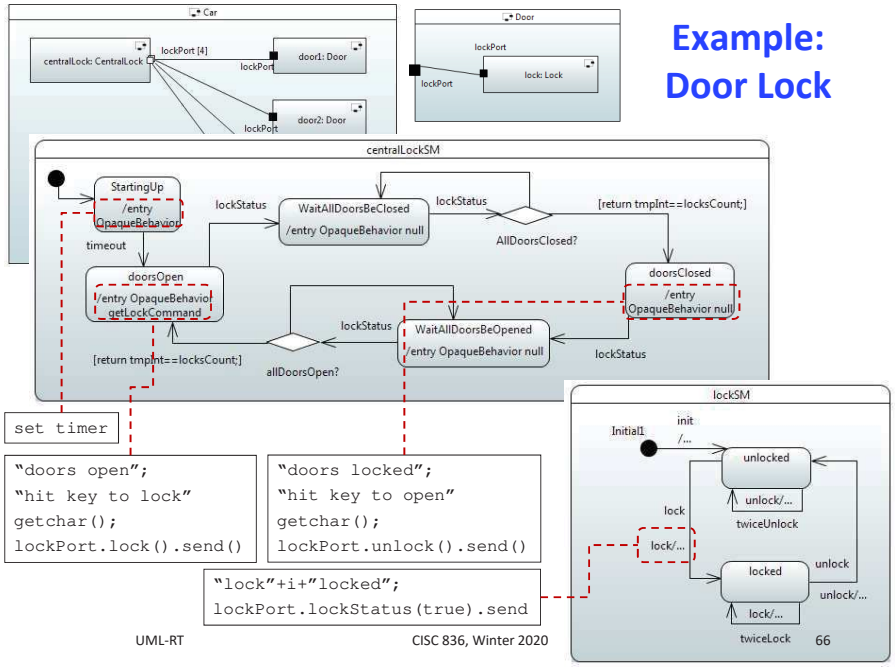


represents n **`Door`** instances each with a single port **`dPort`** that connects them to one of the n instances of **`cPort`**, where 2≤n≤4. E.g.,



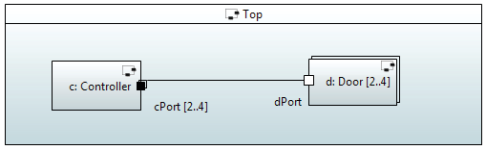
  - Replication ranges [m..n] with m<n particularly useful when using dynamic capsule creation (optional and plugin capsules)

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 error message
    - 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)

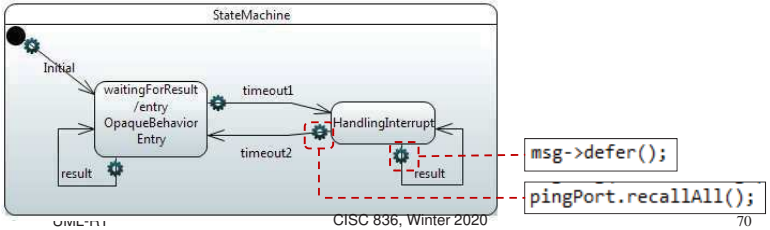
- **`RTMessage`**
  - Base type for messages
    - Created upon send signal event; refers to signal being sent and its 'payload'
    - Signals separated from messages, so that different messages can refer to same signal (for broadcast signals)
  - Methods
    - **`bool defer()`**
      - Put message 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

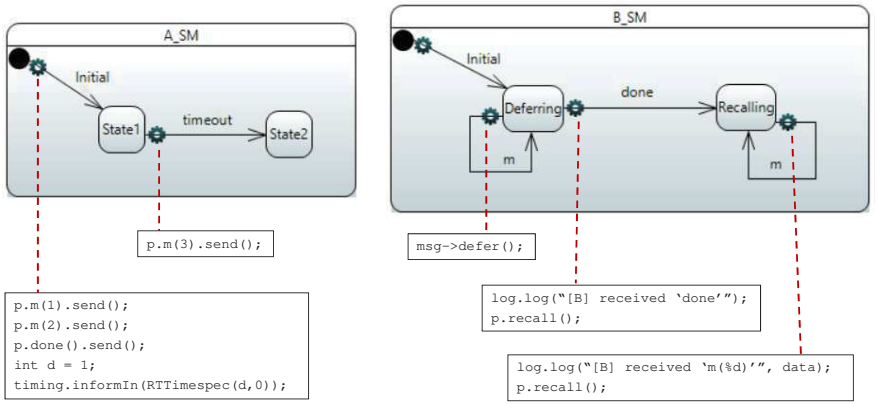
- Allows handling of messages that arrive while in 'wrong' state
- Defer m message on port p:
  - 'Wrong' state has self transition triggered by m with effect **`'msg->defer()'`**
- Recall message m on port p:
  - When entering state in which m should be handled, execute **`'p.recall()'`**, **`'p.recallAll()'`**,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

Port	Protocol Message
pingPort	in result



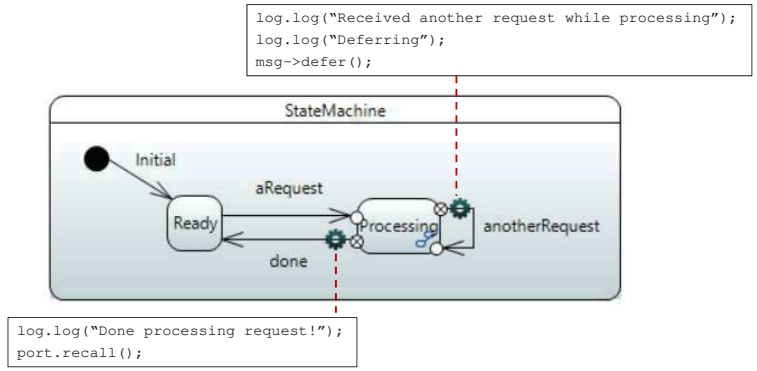
Defer/recall (Cont'd)

- 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();`**

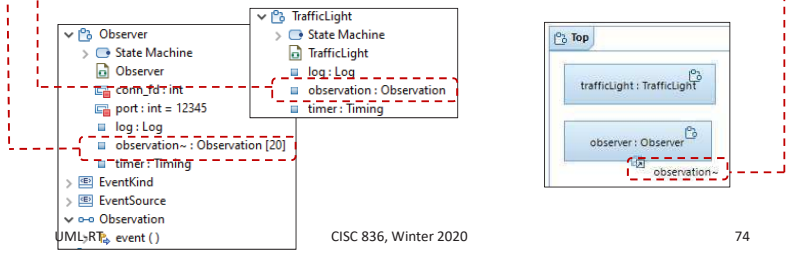


Run Time Services (RTS) Library: Communication (3)

- **`RTProtocol`**
  - Base type for protocols
  - Methods
    - **`bool recall()`**
      - Move matching message from one instance (port) from defer queue to end of message queue
    - **`bool recallAll()`**
      - Move matching messages from all instance (ports) from defer queue to end of message queue
    - **`bool recallFront()`**
      - Move matching message from one instance (port) from defer queue to front of message queue

Dynamic Change II: Wired Ports

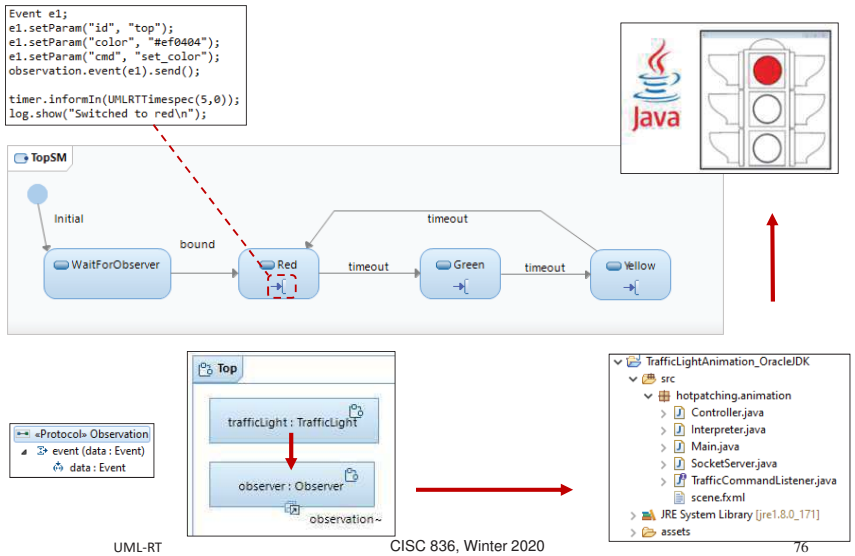
- So far, only wired ports
  - Connected automatically when instances are created
- Unwired ports
  - Connected at run-time via 'publish/subscribe'
    - Port on publisher: Service Provision Point (registration kind = automatic, conjugated)
    - Port on subscriber: Service Access Point (registration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



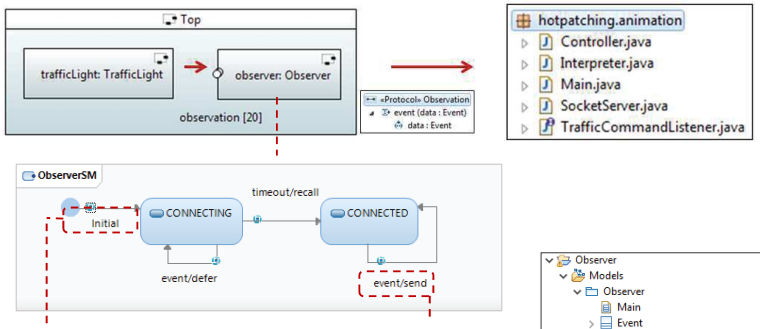
Run Time Services (RTS) Library: Communication (3)

- **`RTProtocol`**
  - Base type for protocols
  - Methods
    - **`bool recall()`**
      - Move matching message from one instance (port) from defer queue to end of message queue
    - **`bool recallAll()`**
      - Move matching messages from all instance (ports) from defer queue to end of message queue
    - **`bool recallFront()`**
      - Move matching message from one instance (port) from defer queue to front of message queue
    - **`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
      - Typically, one SPP port and multiple SAP ports
    - **`bool deregisterSAP(), bool deregisterSPP()`**

How Does the Observer Work?



How Does the Observer Work? (Cont'd)



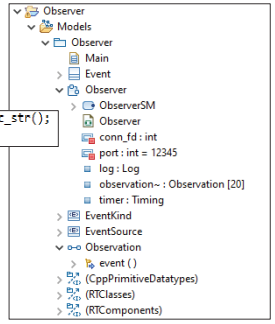
Initialize socket connection

```
int ret = 0;
struct sockaddr_in server_addr = { 0 };
server_addr.sin_family = AF_INET;
server_addr.sin_port = htons(this->port);
ret = inet_pton(AF_INET, "127.0.0.1", &server_addr.sin_addr);

this->conn_fd = socket(AF_INET, SOCK_STREAM, 0);
if(this->conn_fd < 0) {
    log.log("[Observer] error creating socket");
} else {
    ret = connect(this->conn_fd, (struct sockaddr*)&server_addr, sizeof(server_addr));
    if(ret == -1)
        log.log("[Observer] error connecting to host");
    else
        timer.informIn(RTTimespec(0, 0));
}
```

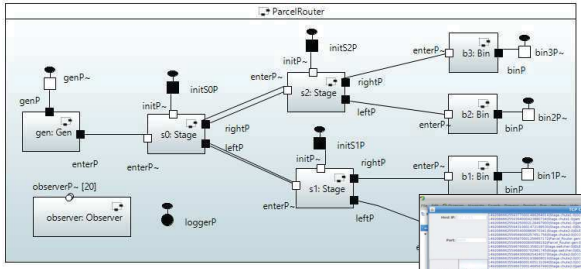
Write to socket

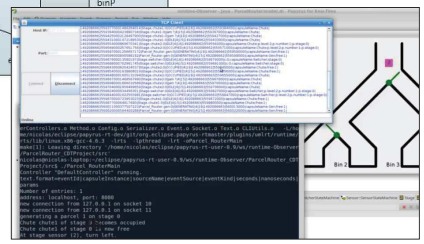
```
const char* eventStr = ((Event*)rtdata)->serialize().c_str();
send(this->conn_fd, eventStr, strlen(eventStr), 0);
```



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 RSARTE

- 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
  - with or without, e.g.,
    - animation
    - advanced UML-RT features (plugin capsules, inheritance, multi-threading, priorities)
    - advanced RSARTE features (e.g., tracing and debugging [Moh10f, HCL20a])
    - external components (C++ libraries, analytics, learning)
  - to
    - illustrate or explore (e.g., basic or advanced UML-RT or RSARTE features, use of external components, fault-tolerance or adaptation technique)

Project Suggestions II: Extensions of UML-RT

- Explore HCL extensions and sample systems
  - TCP, HTTP, IoT, https://github.com/hcl-pnp-rtist
- UML-RT for IoT and distributed systems
  - Queen's Papyrus-RT extensions
  - https://github.com/kjahed/Models18-MMCA/blob/master/index.md
- Explore and model the RTS
  - How much of the structure and behavior of the RTS can we capture using plain class and sequence diagrams?
- Sequence diagram generation
  - Instrument model (or code) such that at runtime text is written to a file that describes the execution as sequence diagram in, e.g., the textual notation of PlantUML (https://plantuml.com/sequence-diagram)

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
  - 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, Orleans, …
- ?
  - Talk to me

Project Proposal due Thurs, March 5