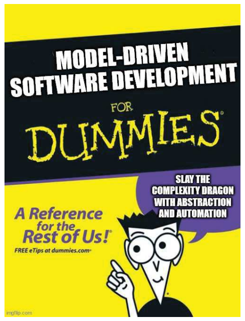
CISC836: Models in Software Development: Methods, Techniques and Tools


MODEL-DRIVEN SOFTWARE DEVELOPMENT
FOR
DUMMIES
SLAY THE COMPLEXITY DRAGON WITH ABSTRACTION AND AUTOMATION
A Reference for the Rest of Us!
FREE eTips at dummies.com

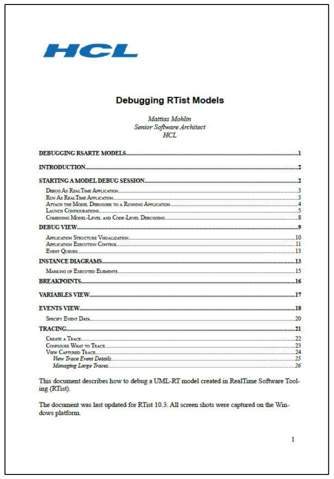
UML-RT and RSARTE: Part V

Juergen Dingel
Oct 2021

UML-RT/RSARTE: Part V

- Model debugger
- UML-RT: other features
  - Inheritance
    - Capsule, state machine
  - Synchronous calls
    - invoke vs send in `RTProtocol.h`
  - Message priorities
- Generating multi-threaded code
- Support for distributed, web-based systems

Debugging in RSARTE



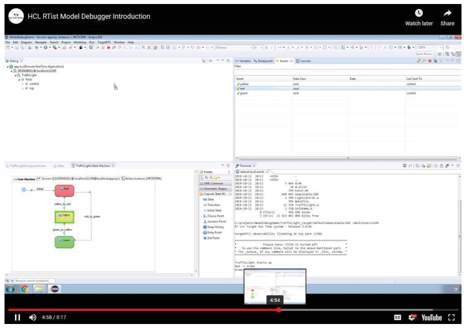
Debugging RTist Models

Mattias Mohlin
Senior Software Architect
HCL

This document describes how to debug a UML-RT model created in RealTime Software Tooling (RTist).

The document was last updated for RTist 10.3. All screen shots were captured on the Windows platform.

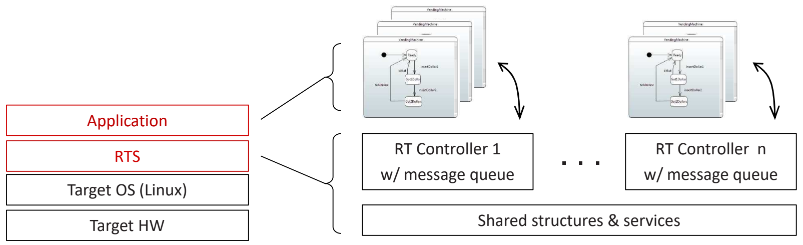
1



https://www.youtube.com/embed/_oeJgrMb3UU

https://rtist.hcldoc.com/help/topic/com.ibm.xtools.rsarte.webdoc/pdf/RTist%20Model%20Debug.pdf

Creating Multi-Threaded Applications


Application
RTS
Target OS (Linux)
Target HW
RT Controller 1 w/ message queue
. . .
RT Controller n w/ message queue
Shared structures & services

- Fixed capsule parts
  - instance always runs in the same thread as owning capsule instance
- Optional and plugin capsule parts
  - Instance can run in its own physical thread
- Each physical thread
  - has one controller w/ its own message queue, executing possibly many state machines

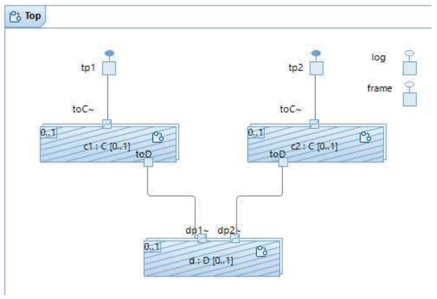
Creating Multi-Threaded Applications (Cont'd)

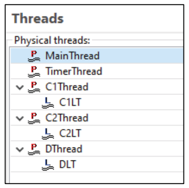
- Logical thread
  - Refers to the execution of a capsule instance/state machine
- To make the instance in optional capsule part c1 run in its own thread:

  In transformation configuration:
  1. Create physical thread w/ some name, e.g., 'PTc1'
  2. Create logical thread w/ some name 'LTc1'

     => Code generator creates variable 'RTController LTc1'
  3. Assign LTc1 to PTc1

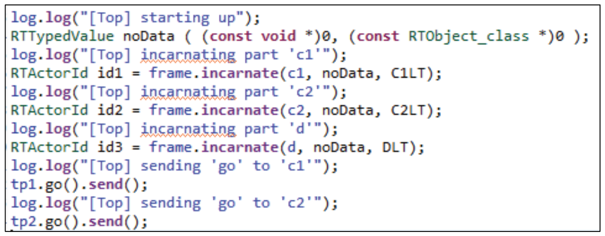
  In capsule owning c1:
  1. when incarnating c1, use special version of **incarnate** with 'LTc1' as argument



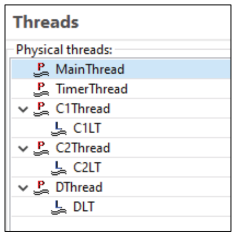

Threads
Physical threads:
- MainThread
- TimerThread
- C1Thread
  - C1LT
- C2Thread
  - C2LT
- DThread
  - DLT

Creating Multi-Threaded Applications (Cont'd)

- Example

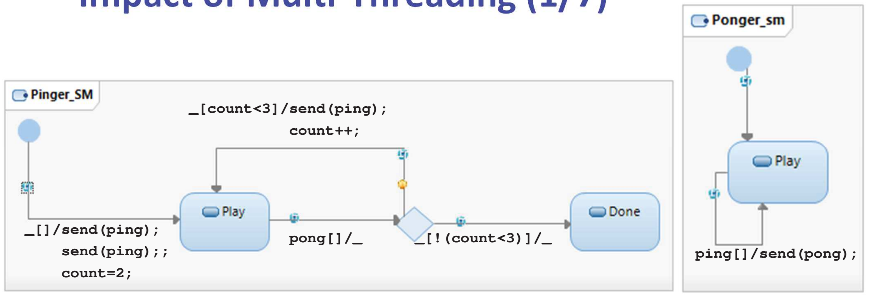


```
log.log("[Top] starting up");
RTTypedValue noData ( (const void *)0, (const RTObject_class *)0 );
log.log("[Top] incarnating part 'c1'");
RTActorId id1 = frame.incarnate(c1, noData, C1LT);
log.log("[Top] incarnating part 'c2'");
RTActorId id2 = frame.incarnate(c2, noData, C2LT);
log.log("[Top] incarnating part 'd'");
RTActorId id3 = frame.incarnate(d, noData, DLT);
log.log("[Top] sending 'go' to 'c1'");
tp1.go().send();
log.log("[Top] sending 'go' to 'c2'");
tp2.go().send();
```

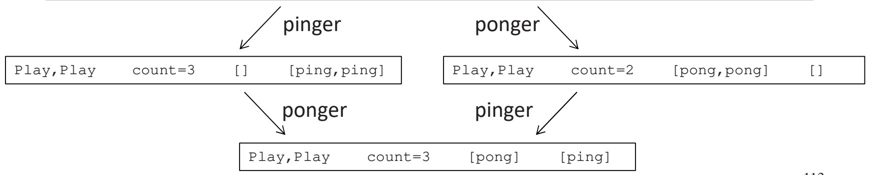

Threads
Physical threads:
- MainThread
- TimerThread
- C1Thread
  - C1LT
- C2Thread
  - C2LT
- DThread
  - DLT

- Incarnation w/ thread assignment (**RTFrame**)
  - **RTActorId incarnate (RTActorRef & cp, RTypedValue & info, RTController * controller, int index)**
    - **info** is data to be passed into incarnated part
    - **controller** is controller which should run the incarnated part
    - **index** specifies where to insert part in case of replicated parts

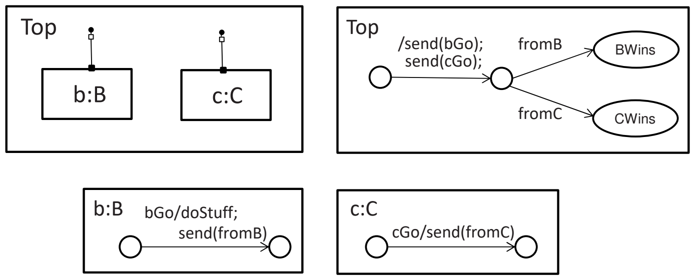
Impact of Multi-Threading (1/7)



actS	vars	mq1 (pinger thread)	mq2 (ponger thread)
_,_	count=_	[]	[]
Play,Play	count=2	[]	[ping,ping]
		[pong]	[ping]


pinger
ponger
Play,Play count=3 [] [ping,ping]
Play,Play count=2 [pong,pong] []
ponger
pinger
Play,Play count=3 [pong] [ping]

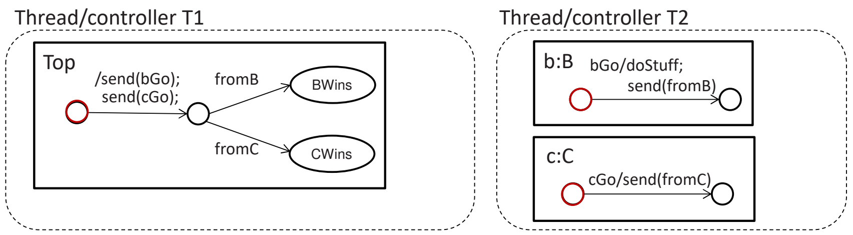
Impact of Multi-Threading (2/7)



- What if b and c execute on
  - the same thread/controller (and, thus, share a message queue)?
  - different threads/controllers (and, thus, have their own message queues)?
- What if 'doStuff' takes a really long time?
- What about 'run-to-completion'?

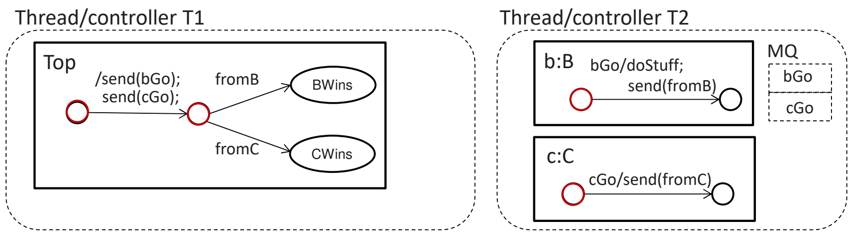
Impact of Multi-Threading (3/7)

b and c run on **same** thread



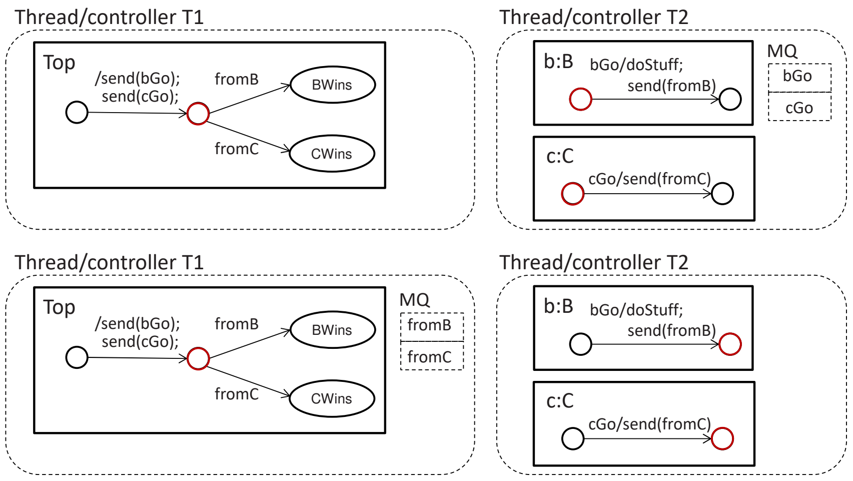
Impact of Multi-Threading (4/7)

b and c run on **same** thread



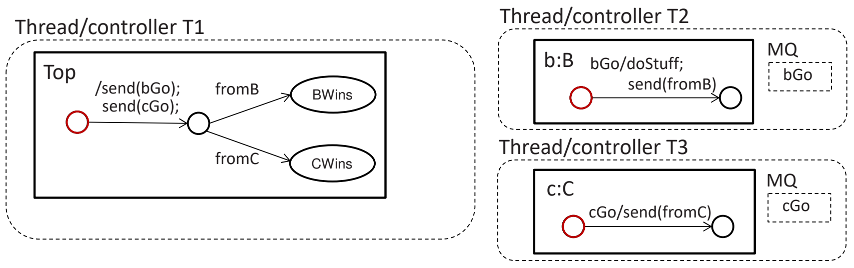
Impact of Multi-Threading (5/7)

b and c run on **same** thread



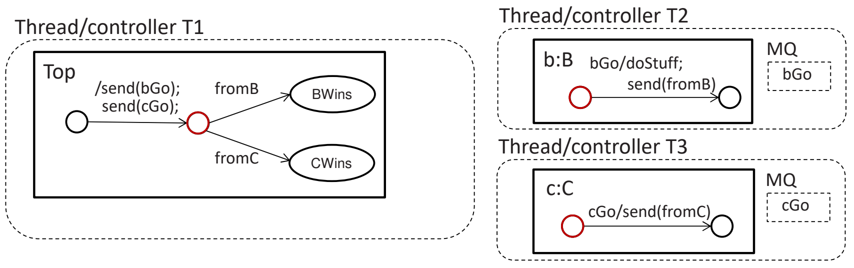
Impact of Multi-Threading (6/7)

b and c run on **different** threads



Impact of Multi-Threading (7/7)

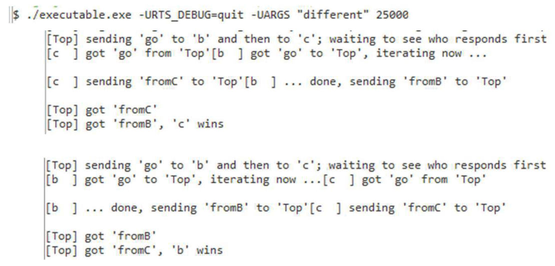
b and c run on **different** threads



3 cases:

if doStuff 'short', b always wins
elsif doStuff 'long', c always wins
else ?

Model available as sample model



```
$ ./executable.exe -URTS_DEBUG=quit -UARGS "different" 25000
    [Top] sending 'go' to 'b' and then to 'c'; waiting to see who responds first
    [c  ] got 'go' from 'Top'[b  ] got 'go' to 'Top', iterating now ...

    [c  ] sending 'fromC' to 'Top'[b  ] ... done, sending 'fromB' to 'Top'

    [Top] got 'fromC'
    [Top] got 'fromB', 'c' wins


    [Top] sending 'go' to 'b' and then to 'c'; waiting to see who responds first
    [b  ] got 'go' to 'Top', iterating now ...[c  ] got 'go' from 'Top'

    [b  ] ... done, sending 'fromB' to 'Top'[c  ] sending 'fromC' to 'Top'

    [Top] got 'fromB'
    [Top] got 'fromC', 'b' wins
```

Creating Multi-Threaded Applications (Cont'd)

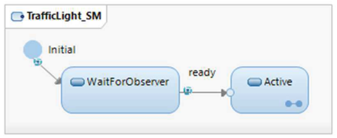
- Pros
  - Make parts of application more independent
    - Long execution steps in one part will not reduce responsiveness of another
  - If threads have priority
    - better performance for tasks on threads with higher priority
  - If threads are mapped to cores
    - better performance for all tasks
- Cons
  - Multi-threading typically introduces the possibility for more than one transition to be enabled in a stable state configuration

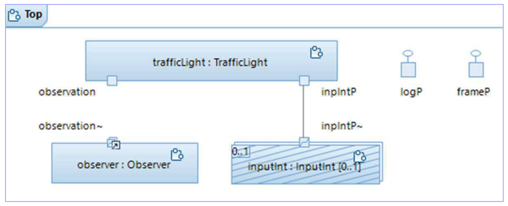
    => Correct design requires ensuring that messages arrive and are processed in correct order by several different controllers
  - Multi-threading makes application more susceptible to
    - Specifics of platform (RTS/OS, hardware) and communication media

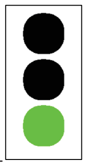
    $\Rightarrow$ Change in RTS, OS, C++ libraries, hardware, resource use can lead to messages being delivered and processed in different order

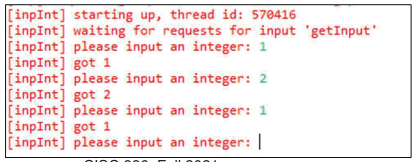
    $\Rightarrow$ Model must be carefully designed to make it robust to these changes

"What if a model needs to receive user input during execution?"





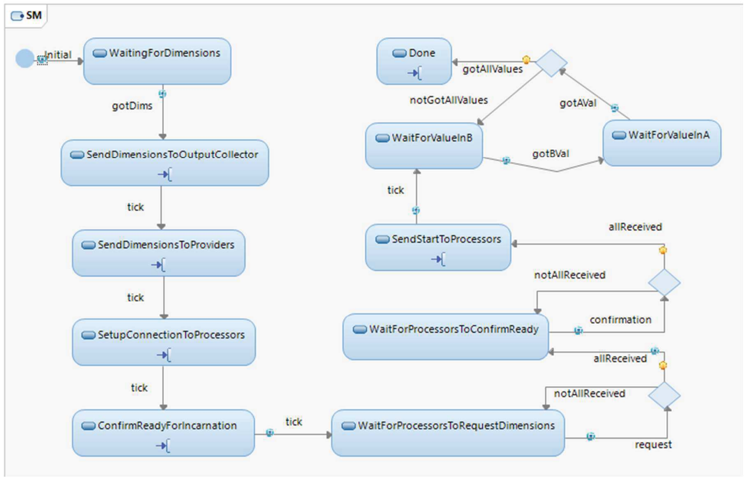




```
[inpInt] starting up, thread id: 570416
[inpInt] waiting for requests for input 'getInput'
[inpInt] please input an integer: 1
[inpInt] got 1
[inpInt] please input an integer: 2
[inpInt] got 2
[inpInt] please input an integer: 1
[inpInt] got 1
[inpInt] please input an integer: |
```

"Every computation needs to be triggered by an incoming message. Isn't that a restriction?"

“Every computation needs to be triggered by an incoming message. Isn’t that a restriction?”


SM
Initial
WaitingForDimensions
gotDims
SendDimensionsToOutputCollector
tick
SendDimensionsToProviders
tick
SetupConnectionToProcessors
tick
ConfirmReadyForIncarnation
tick
WaitForProcessorsToRequestDimensions
request
notAllReceived
allReceived
WaitForProcessorsToConfirmReady
confirmation
notAllReceived
allReceived
SendStartToProcessors
tick
WaitForValueInB
gotBVal
WaitForValueInA
gotAVal
notGotAllValues
gotAllValues
Done