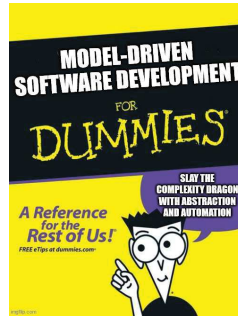


CISC836: Models in Software Development: Methods, Techniques and Tools



UML-RT and RSARTE: Part V

Juergen Dingel
Oct 2021

UML-RT

CISC 836, Fall 2021

107

UML-RT/RSARTE: Part V

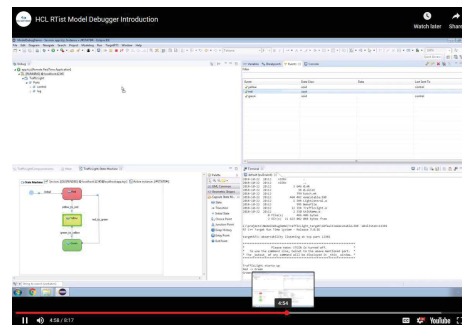
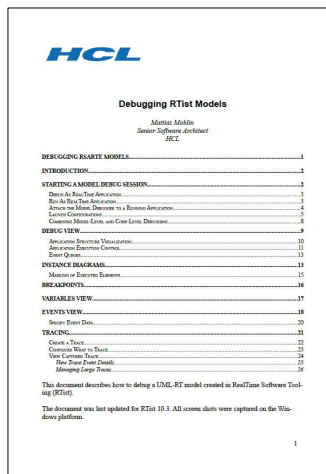
- Model debugger
- UML-RT: other features
 - Inheritance
 - Capsule, state machine
 - Synchronous calls
 - invoke vs send in `RTProtocol.h`
 - Message priorities
- Generating multi-threaded code
- Support for distributed, web-based systems

UML-RT

CISC 836, Fall 2021

108

Debugging in RSARTE



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

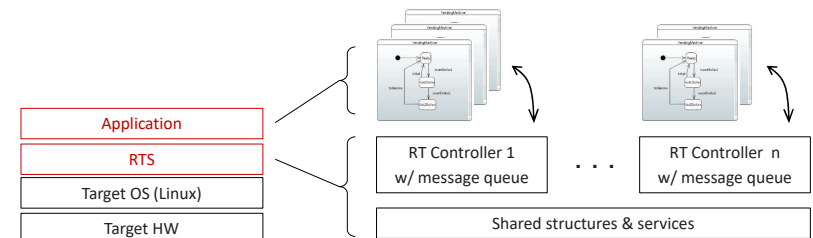
<https://rtist.hcldoc.com/help/topic/com.ibm.xttools.rsarte.webdoc/pdf/RTist%20Model%20Debug.pdf>

UML-RT

CISC 836, Fall 2021

109

Creating Multi-Threaded Applications



- 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

UML-RT

CISC 836, Fall 2021

110

Creating Multi-Threaded Applications (Cont'd)

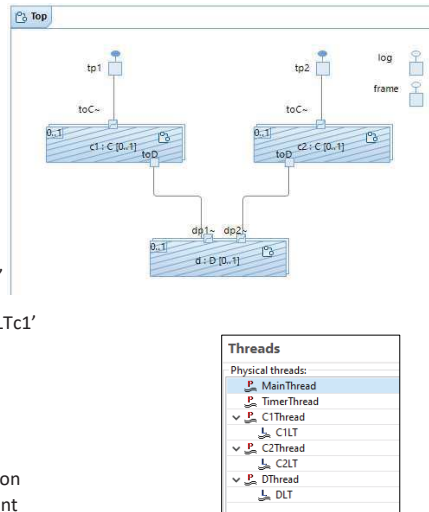
- **Logical thread**
 - Refers to the execution of a capsule instance/state machine
- To make the instance in optional capsule part c1 run in its own thread:

In **transformation configuration**:

 - Create physical thread w/ some name, e.g., 'PTc1'
 - Create logical thread w/ some name 'LTc1'
=> Code generator creates variable 'RTController LTc1'
 - Assign LTc1 to PTC1

In **capsule owning c1**:

 - when incarnating c1, use special version of **incarnate** with 'LTc1' as argument



UML-RT

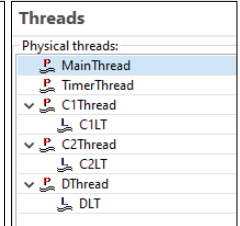
CISC 836, Fall 2021

111

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();
```



■ Incarnation w/ thread assignment (RTFrame)

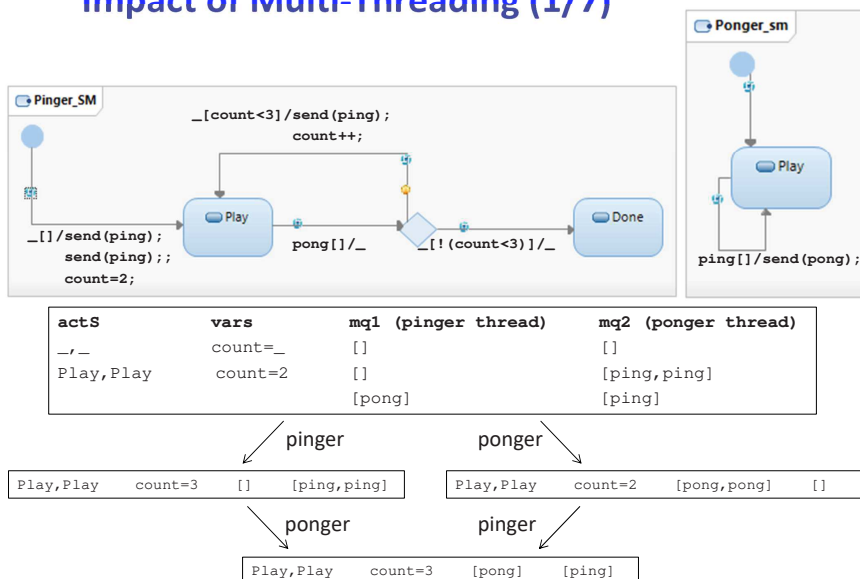
- **RTActorId incarnate (RTActorRef & cp, RTTypedValue & info, RTController * controller, int index)**
 - info** is data to be passed into incarnated part
 - controller** is controller which should run the incarnated part
 - index** specifies where to insert part in case of replicated parts

UML-RT

CISC 836, Fall 2021

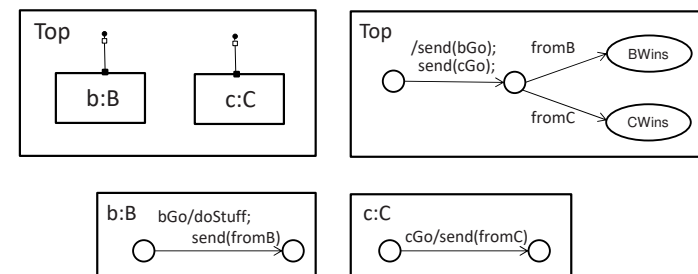
112

Impact of Multi-Threading (1/7)



113

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'?

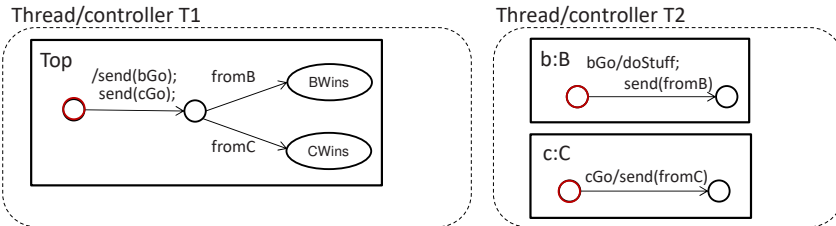
UML-RT

CISC 836, Fall 2021

114

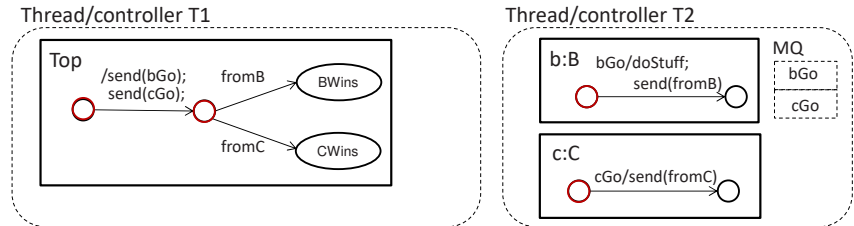
Impact of Multi-Threading (3/7)

b and c run on **same** thread



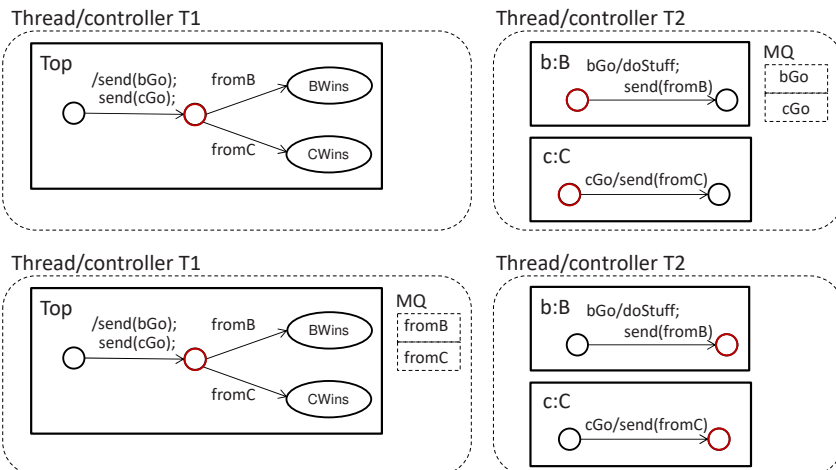
Impact of Multi-Threading (4/7)

b and c run on **same** thread



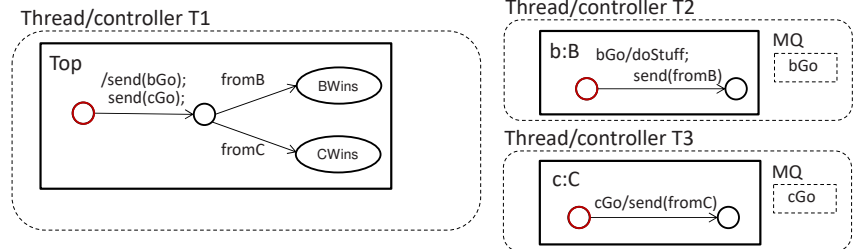
Impact of Multi-Threading (5/7)

b and c run on **same** thread



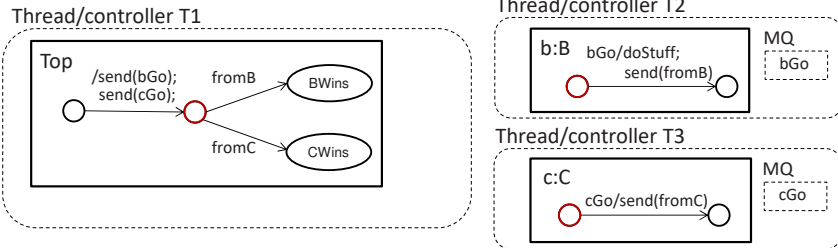
Impact of Multi-Threading (6/7)

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
elseif doStuff 'long', c always wins
else ?
```

Model available as sample model

UML-RT

```
$ ./executable.exe _RTS_DEBUG=quit -UARGS "different" 25000
```

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

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

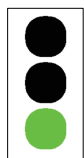
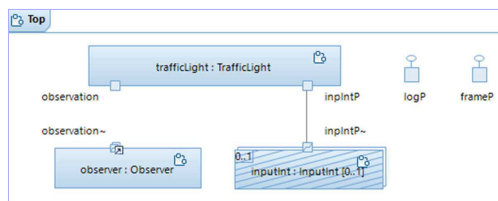
[Top] got 'fromC'
[Top] got 'fromB', 'c' wins
```

```
[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
```

“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: |
```

UML-RT

CISC 836, Fall 2021

121

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

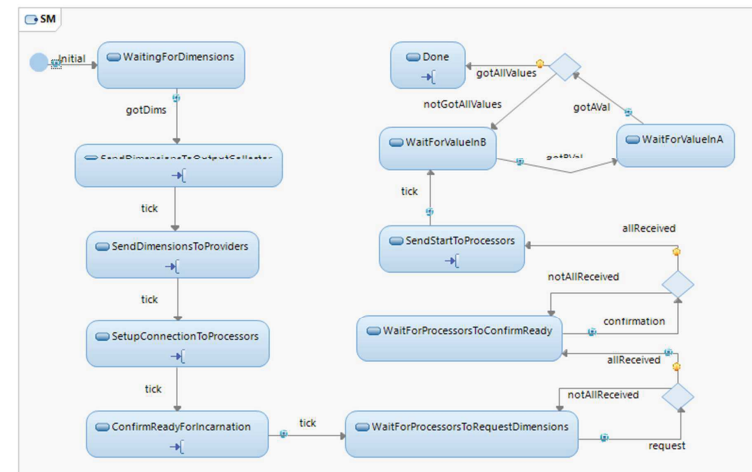
- Multi-threading typically introduces the possibility for more than one transition to be enabled in a stable state configuration
 - => Correct design requires ensuring that messages arrive and are processed in correct order by several different controllers
- Multi-threading makes application more susceptible to
 - Specifics of platform (RTS/OS, hardware) and communication media
 - ⇒ Change in RTS, OS, C++ libraries, hardware, resource use can lead to messages being delivered and processed in different order
 - ⇒ Model must be carefully designed to make it robust to these changes

UML-RT

CISC 836, Fall 2021

120

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



UML-RT

CISC 836, Fall 2021

122