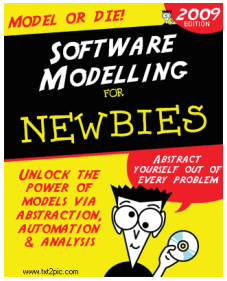
CISC836: Models in Software Development: Methods, Techniques and Tools



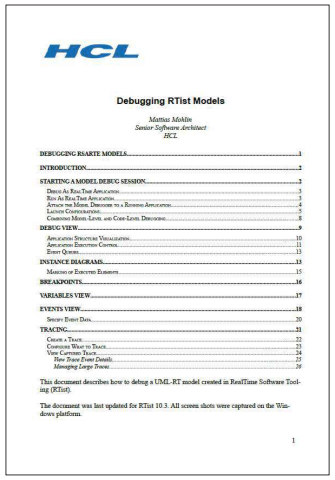
UML-RT and RSARTE: Part V

Juergen Dingel
Feb 2020

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


HCL

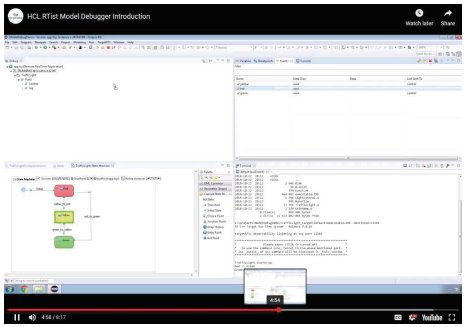
Debugging RTist Models

Mattias Mohlin
Senior Software Architect
IBM.

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.

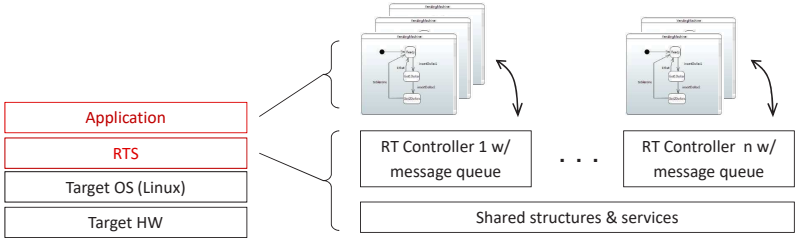
1



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

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

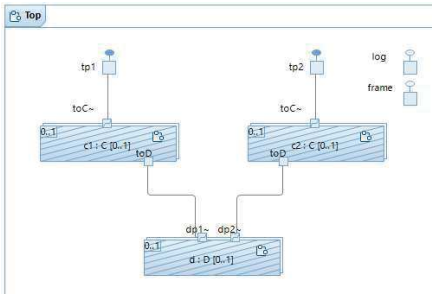
Creating Multi-Threaded Applications



- 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
- Suppose the instance in an optional capsule part c1 is to run in its own thread



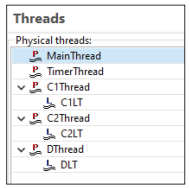
In transformation configuration:

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:

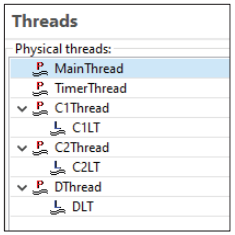
1. when incarnating c1, use special version of **`incarnate`** with 'LTc1' as argument



Creating Multi-Threaded Applications (Cont'd)

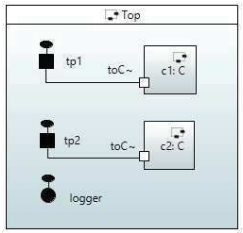
- Example

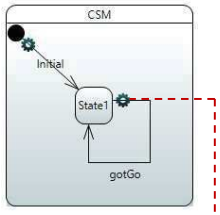
```
log.log("[Top] starting up");
RTTypedValue noData ( (const void *)0, (const RTObject_class *)0 );
log.log("[Top] incarnating part 'c1'");
RTActorId id1 = frame.incarnate(c1, noData, C1LT);
log.log("[Top] incarnating part 'c2'");
RTActorId id2 = frame.incarnate(c2, noData, C2LT);
log.log("[Top] incarnating 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();
```



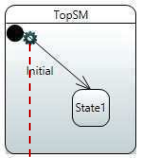
- Incarnation w/ thread assignment
  - **`RTActorId incarnate (RTActorRef & cp, RTypedValue & info, RTController * controller, int index)`**
    - **`info`** is data to be passed into incarnated part
    - **`controller`** is controller which should run the incarnated part
    - **`index`** specifies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

Creating Multi-Threaded Applications (Cont'd)






```
logger.log("[%s] Got 'go'", this->getName());
```




```
logger.log("[Top] Sending 'go' to 'c1'");
tp1.go().send();
logger.log("[Top] Sending 'go' to 'c2'");
tp2.go().send();
```

Consider two scenarios:

1. Top, c1, c2 share one thread
2. Top, c1, c2 each have their own thread

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 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