Model Translation from Papyrus-RT into the nuXmv Model Checker

Ms. Sneha Sahu

2020 September 15

Agenda

Introduction

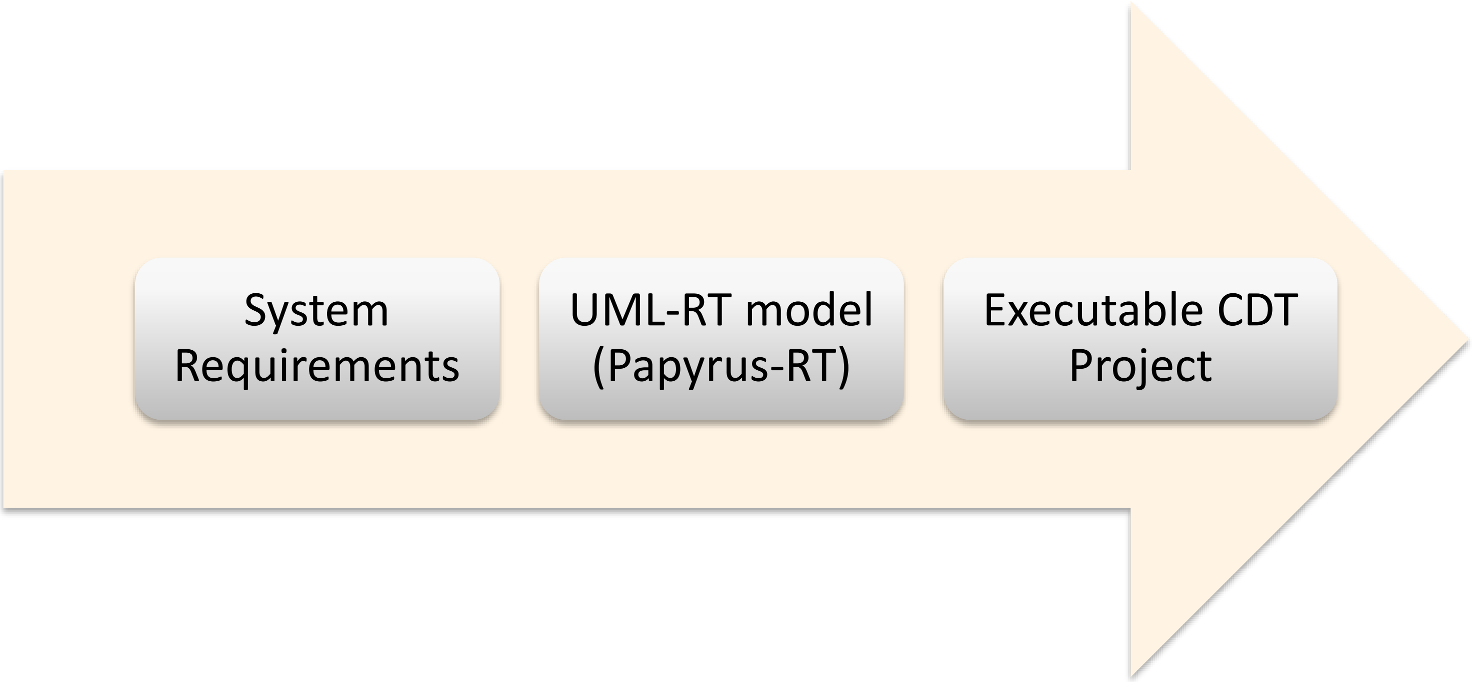
- Overview
- Tools

Translation

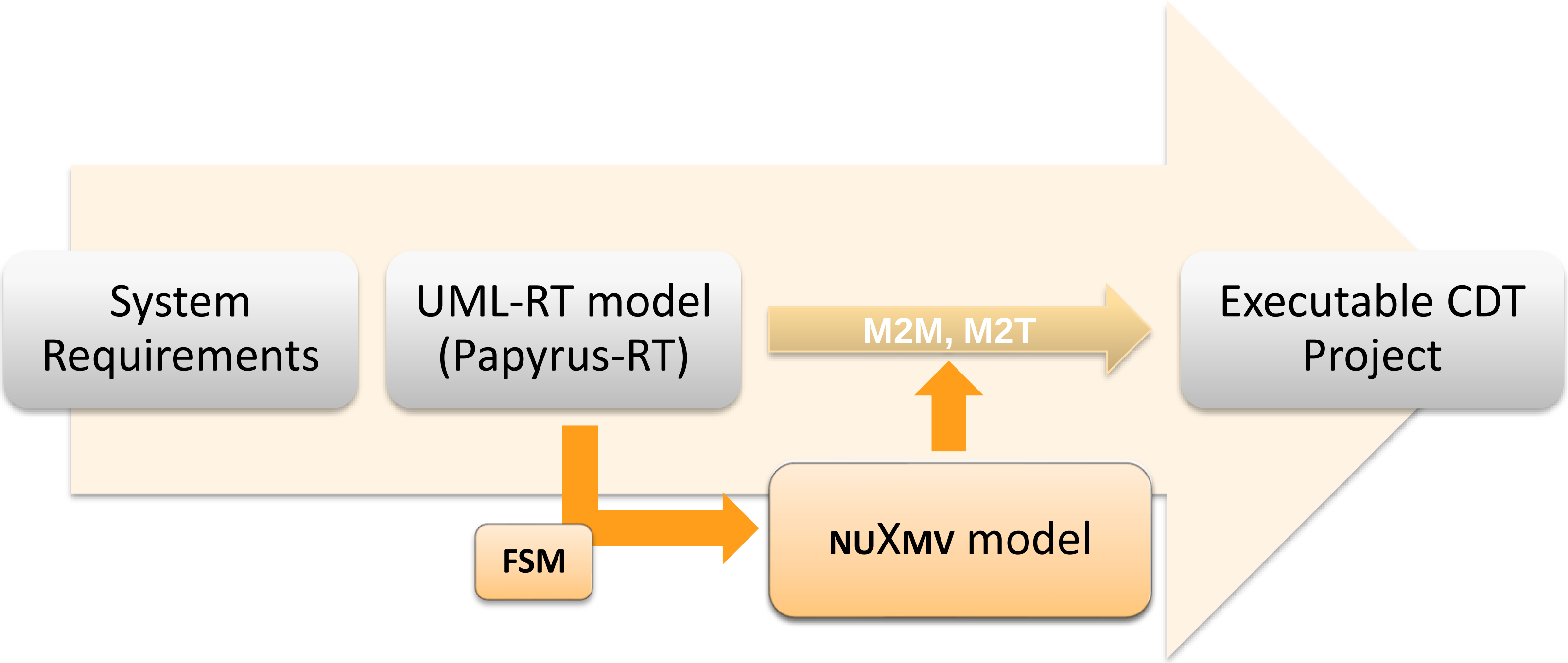
- Approach
- Case Study Analysis

Conclusion

Introduction



Introduction

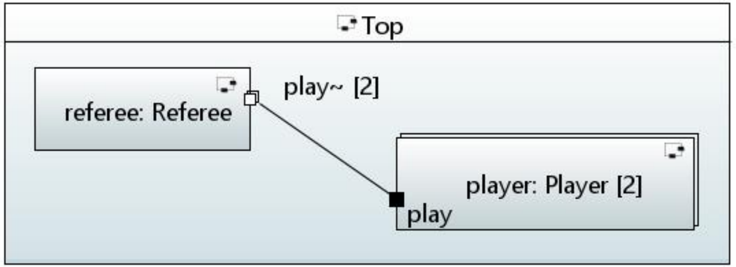


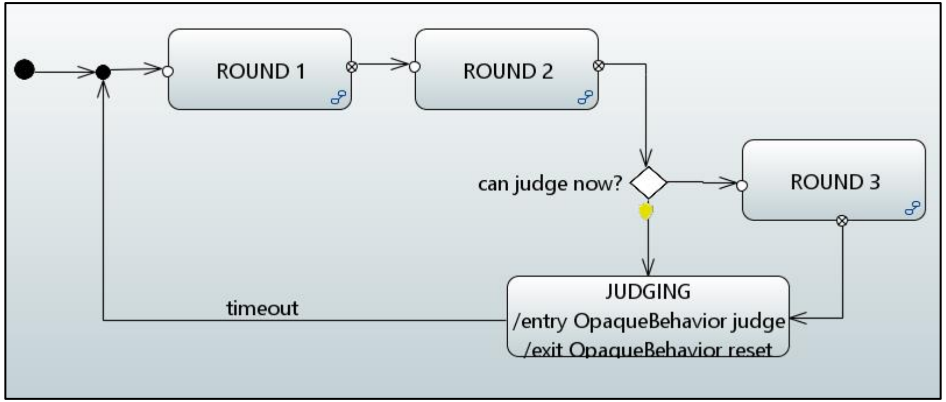
UCA Universidad de Cádiz

FRANKFURT UNIVERSITY OF APPLIED SCIENCES

Papyrus-RT

- Basic Modelling Components – Capsules, Protocols, Ports, Attributes, Triggers, Guards, Composite Structure, State Diagram

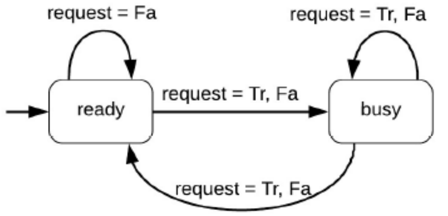




- Port communication via controller message queue
- Run-To-Completion execution semantics

nuXmv Model Checker

```
MODULE main
VAR
  request : {Tr, Fa};
  state : {ready, busy};
ASSIGN
  init(state) := ready;
  next(state) := case
                   state = ready & (request = Tr): busy;
                   TRUE : {ready,busy};
                 esac;
```



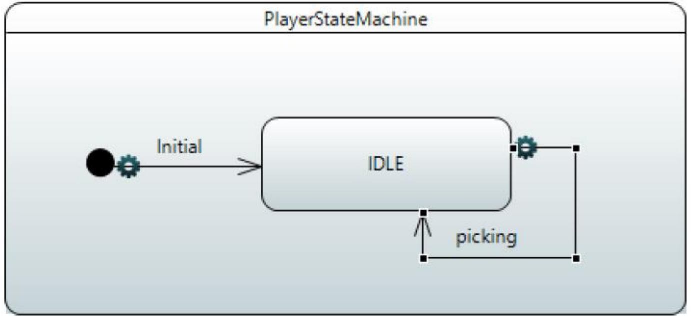
- Module Structure –
  - VARiable declarations
  - INITializations (single assignment constraint)
  - ASSINGments and TRANSitions (circular dependency check)
- Only synchronous systems unlike NuSMV
- BDD and SAT based verification of Finite State Machines
- SMT-based techniques for verification of Infinite State Machines

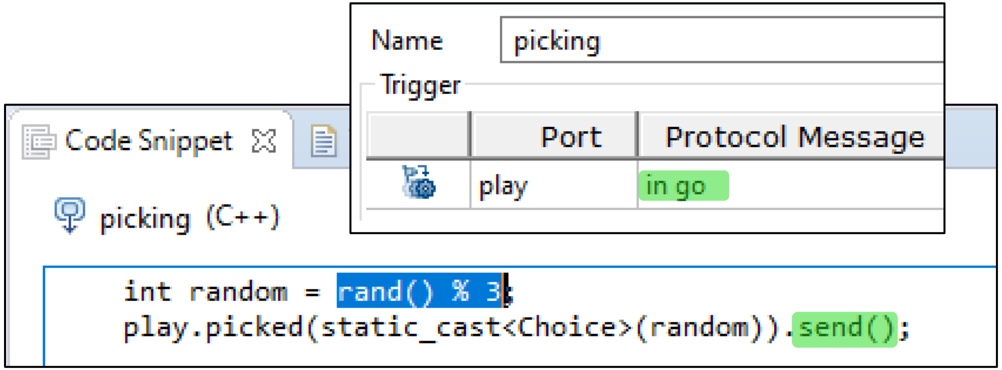
Translation Approach

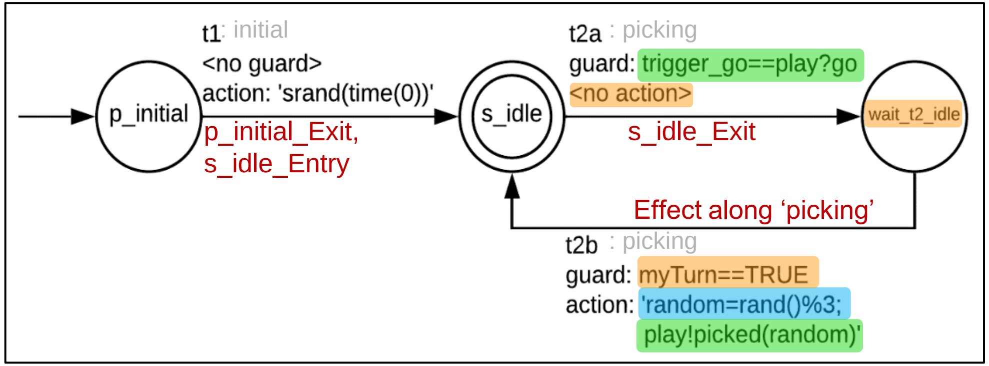
UML-RT	NUXMV
State Diagrams	FSMs
Non trivial transition effects	Abstract out ‘what’, modify ‘how’
Top Capsule	Main Module
Asynchronous Communication*	Controller logic in Main Module
Timing Protocol*	Countdown logic in Separate module
Other Capsules & Protocols	Separate modules with input parameters
Ports	VAR declarations
Connections	as inputs to Modules

** These algorithm are defined based on the gained understanding of the Papyrus-RT system*

SD→ FSM, abstraction








```
ASSIGN
    next(msg_Play):= case
        t2b: picked;
        TRUE: null;
    esac;
    next(param_Play):= case
        t2b: {1, 2, 3};
        TRUE: 0;
    esac;
```

Capsules, Protocols → Modules

- «Capsule» Reservoir
  - Reservoir
  - «RTPort» portLog : Log
  - «RTPort» Res : ~commands
  - «RTPort» InsulinInject : ~insulin
  - rLvl : unsigned int
  - rLvl_max : unsigned int
    - 100
  - rLvl_min : unsigned int
  - rLB : unsigned int
  - inputDose : unsigned int
    - <Comment> min = 0
    - <Comment> max = 3

- «Protocol» insulin
  - out ask (doseComputed : unsigned int)
    - doseComputed : unsigned int
  - in inject (doseGiven : unsigned int)
    - doseGiven : unsigned int

```
MODULE main
 VAR
   ...
   reservoir: capsule_reservoir(TURN_RES, port_button,
   bgc.port_insulinAsk);
   ...
```

```
MODULE capsule_reservoir(myTurn, in_res, in_insulinInject)
 VAR
   port_res: protocol_commands(TRUE, msg_res, 0);
   msg_res: {null, fail};
   port_insulinInject: protocol_insulin(TRUE, msg_insulinInject,
    param_insulinInject);
   msg_insulinInject: {null, inject};
   param_insulinInject: 0..3;
   rLvl: const_rLvl_min .. const_rLvl_max;
   state: {p_initial, s_start, s_ok, wait_t3_ok, wait_t4_ok,
   wait_t5_ok, s_stop};
   inputDose: 0..3;
 DEFINE
   WAITING := (state = wait_t3_ok | state = wait_t4_ok | state =
    wait_t5_ok);
   const_rLvl_max := 100;
   const_rLvl_min := 0;
```

Capsules, Protocols → Modules

```
«Capsule» Reservoir
    Reservoir
    «RTPort» portLog : Log
    «RTPort» Res : ~commands
    «RTPort» InsulinInject : ~insulin
  rLvl : unsigned int
  rLvl_max : unsigned int
      100
  rLvl_min : unsigned int
  rLB : unsigned int
  inputDose : unsigned int
      <Comment> min = 0
      <Comment> max = 3
```

```
MODULE main
 VAR
    ...
    reservoir: capsule_reservoir(TURN_RES, port_button,
    bgc.port_insulinAsk);
    ...
```

```
MODULE capsule_reservoir(myTurn, in_res, in_insulinInject)
 VAR
    port_res: protocol_commands(TRUE, msg_res, 0);
    msg_res: {null, fail};
    port_insulinInject: protocol_insulin(TRUE, msg_insulinInject,
     param_insulinInject);
    msg_insulinInject: {null, inject};
    param_insulinInject: 0..3;
    rLvl: const_rLvl_min .. const_rLvl_max;
    state: {p_initial, s_start, s_ok, wait_t3_ok, wait_t4_ok,
```

```
«Protocol» insulin
  out ask (doseComputed : unsigned int)
    doseComputed : unsigned int
  in inject (doseGiven : unsigned int)
    doseGiven : unsigned int
```

```
MODULE protocol_insulin(conjugated, msg, param)
 DEFINE
    out_ask := (conjugated = FALSE & msg = ask);
    out_ask_doseComputed := (out_ask ? param : 0);
    in_inject := (conjugated = TRUE & msg = inject);
    in_inject_doseGiven := (in_inject ? param : 0);
```

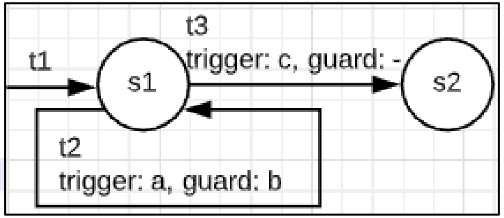
Additional rules to handle '**empty FSM**' situation

[Consider an FSM with 3 transitions – t1, t2, t3]

- (*less than zero case*)
  An extra transition 't_none' is defined, to take care of any undefined transition.

```
DEFINE t_none:= !(t1|t2|t3); TRANS t_none -> next(*) = *;
```

- (*more than 1 case*)
  Avoid non-deterministic **FSM**, by ensuring mutual exclusion of all outgoing transitions from any particular state



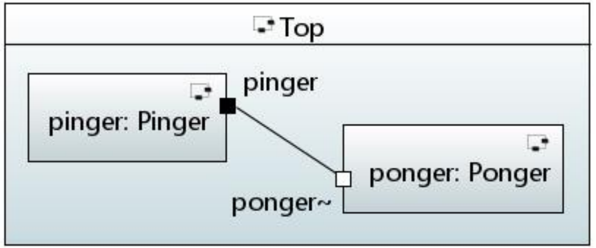

```
t2:= (state = s1 & trigger_a & guard_b);
t3:= (state = s1 & trigger_c);
--issue: non detreminism if both trigger_a and trigger_c are true simultaneously
t2:= (state = s1 & !trigger_c & trigger_a & guard_b);
t3:= (state = s1 & trigger_c);
```

- Add an INVARiant to check that exactly one transitions is valid at any instance.

```
INVAR ( t_none | (t1 & !t2 & !t3) | (!t1 & t2 & !t3) | (!t1 & !t2 & t3) )
```

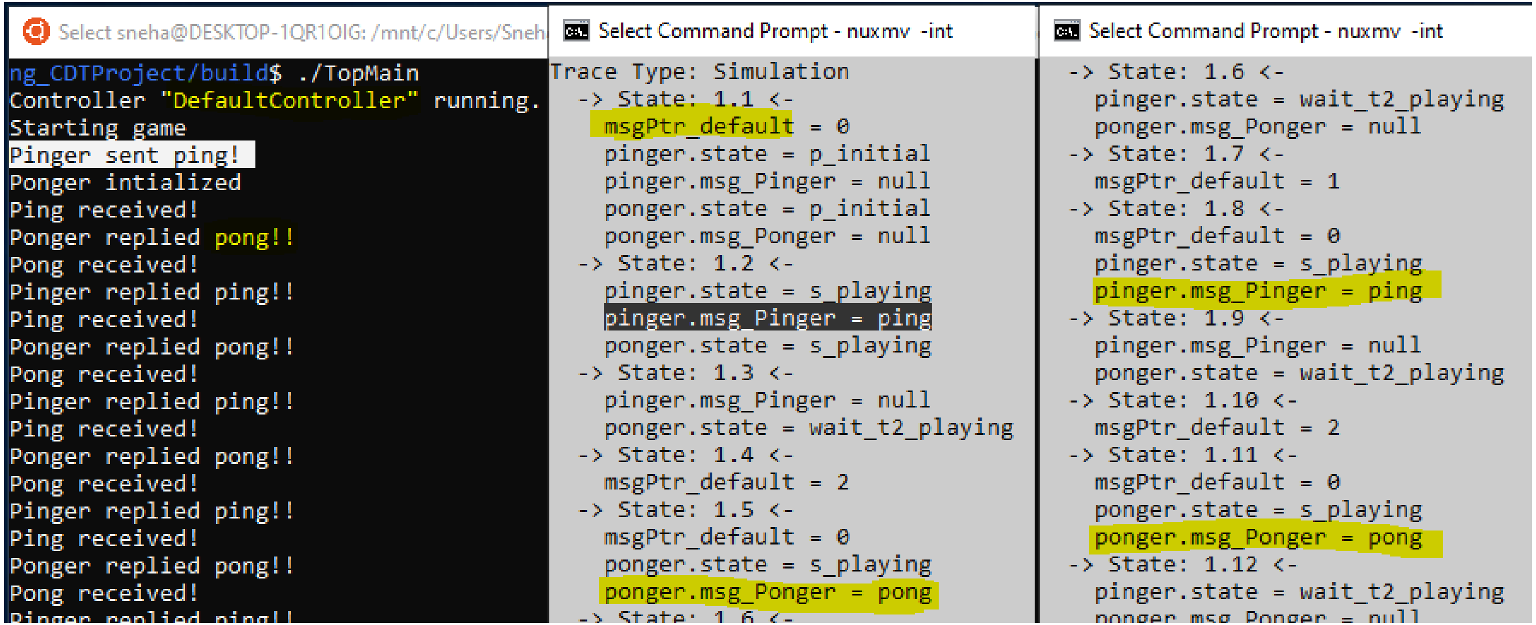
Ping Pong Model

- Single controller queue, with 2 units, communicating directly
- Only one message in queue at any instance of time
- Complexities: none



- Properties Verification –
  - "A 'ping' must be followed by a 'pong' and a 'pong' must be followed by a 'ping'"
    SPEC AG (MSG_PING -> **A** [**AX** !MSG_PING **U** MSG_PONG] & **AF** MSG_PONG)
  - "communication starts with a 'ping'"
    SPEC A [(!MSG_PING & !MSG_PONG) **U** MSG_PING]

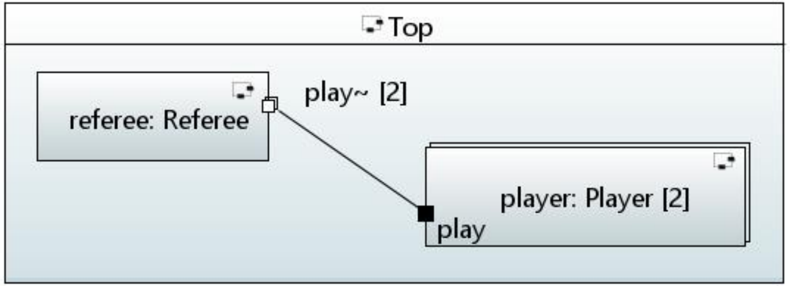
Ping Pong Model



Papyrus-RT nuXmv

Rock Paper Scissor Model

- Single controller queue, with 3 units
- Possibility of more than one message in queue at any instance of time
- Complexities:
  - Multiplicity of capsules and ports
  - Connections carry payload
  - Hierarchical State Diagram
  - Non-trivial code snippets



Rock Paper Scissor Model

- Properties Verification – each Player:
  - “a `player’ always eventually waits in ‘idle’ mode for a signal from the `referee' to be able to play its round”
    SPEC AG AF (**A** [state=s_idle **U** inPlay.in_go])
  - “Whenever a player receives a 'go', it eventually sends out a choice from the given domain and goes back to 'idle’”
    SPEC AG (inPlay.in_go →
    AF (param_Play in {1,2,3} → msg_Play=picked & state=s_idle))

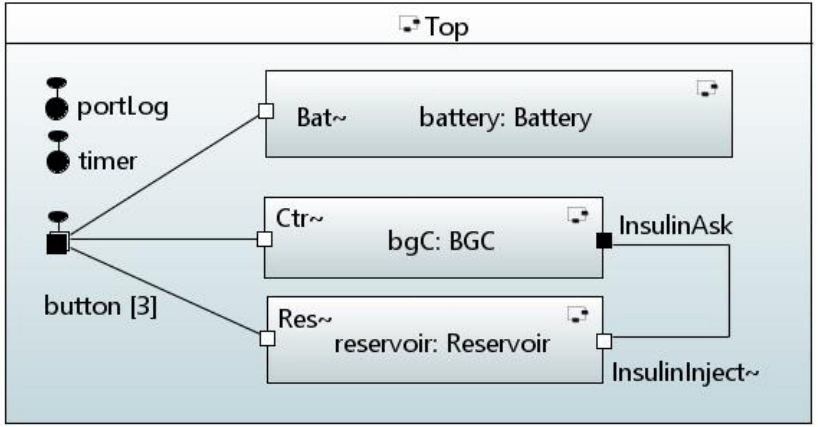
```
-> State: 1.8 <-
  player1.state = wait_t2_idle
  player2.state = wait_t2_idle
  referee.msg_Play = null
  referee.param_Play = 0
```

```
-> State: 1.9 <-
  msgPtr_default = 1
-> State: 1.10 <-
  msgPtr_default = 0
  player1.msg_Play = picked
  player1.param_Play = 1
  player1.state = s_idle
```

```
-> State: 1.11 <-
  msgPtr_default = 2
  ...
-> State: 1.12 <-
  msgPtr_default = 0
  player2.msg_Play = picked
  player2.param_Play = 2
  player2.state = s_idle
```

Insulin Pump Model

- *Multiple* controller queue, with 4 units
- Possibility of more than one message in queue at any instance of time
- Complexity:
  - Multiplicity of ports
  - Connections with/without payload
  - Multi-threading
  - Non-trivial code snippets
  - Multiple overlapping timer reset requests



Insulin Pump Model

- Properties Verification – BG controller unit:
  - “The injected insulin dose should always be within the prescribed maximum limit”
    SPEC AG (inputDose <= const_maxAllowedDose**)**
  - “injected dose is same as the requested amount”
    LTLSPEC G in_insulinAsk.in_inject → prop_drugAsked_equals_drugToGive
  - “when the blood glucose is in the normal or lower range, computed dose must always be zero”
    SPEC AG ((prop_bg_low | prop_bg_normal) → computedDose = 0)

Insulin Pump Model

- Properties Verification – system:
  - “a `Running' system should eventually succeed in keeping blood glucose within the normal range”
    CTLSPEC AG (prop_system_running → AF (bgc.prop_bg_normal | prop_system_stopped))
  - “system stops only when either battery or reservoir fail”
    SPEC AG prop_system_running → **A** [**AX** prop_system_running **U** (!battery.prop_ok | !reservoir.prop_ok)]
  - “when system is stopped, ensure all modules stop eventually and remain so”
    SPEC AG (prop_system_stopped → **AF AG** (battery.prop_stopped & reservoir.prop_stopped & bgc.prop_stopped))

Insulin Pump Model

Select sneha@DESKTOP-1QR1OIG: /mnt/c/Users/Sneha/workspace-papyru

```
-- Initialized >> Reservoir
## starting Reservoir
   # insulin level: 25
-- Initialized >> Blood Glucose Controller (bgc)
## starting bgc
   #      Blood Glucose: previous = 0, current = 155
-- Initialized >> Battery
## starting Battery
   # battery status: 30
   # battery status: 29

 ...

  #      Blood Glucose: previous = 130, current = 130
  # battery status: 17
  # battery status: 16
    > 'bgc' sending 'insulin.ask(1)' --> 'reservoir'.
    # 'reservoir' recieved 'insulin.ask(1)'.
```

Command Prompt - nuxmv -int

```
-> State: 2.1 <-
  ...
  battery.bLvl = 30
  battery.state = p initial
  ...
  reservoir.rLvl = 25
  reservoir.state = p_initial
  ...
  bgc.state = p_initial
  bgc.bgLvl = 155
  ...
  STATE = p_initial
-> State: 2.2 <-
  battery.state = s_start
  reservoir.state = s_start
  bgc.state = s_start
  msg_button = start
  STATE = s_running
```

Insulin Pump Model

```
#      Blood Glucose: previous = 130, current = 130
# battery status: 17
# battery status: 16
  > 'bgc' sending 'insulin.ask(1)' --> 'reservoir'.
  # 'reservoir' recieved 'insulin.ask(1)'.
  > 'reservoir' sending 'inject(1)' --> 'bgc'.
  > 'reservoir' sending 'fail' --> ... !!
  # 'bgc' recieved 'insulin.inject(1)'.
#      Blood Glucose: previous = 130, current = 125
> Stopping Pump
  > ... sending 'stop' --> 'battery'.
  > ... sending 'stop' --> 'reservoir'.
  > ... sending 'stop' --> 'bgc'.
# insulin level: 19
  # 'reservoir' recieved 'stop'.
- Reservoir Stopped [19 units]
  # 'battery' recieved 'stop'.
- Battery Stopped [16 units]
  # 'bgc' recieved 'stop'.
- BGC Stopped [BG = 125 units]
```

```
  battery.state = s_start
  reservoir.state = s_start
  bgc.state = s_start
  msg_button = start
  STATE = s_running
```

```
-> State: 2.118 <-
  ...
  battery.bLvl = 11
  battery.state = s_stop
  ...
  reservoir.rLvl = 19
  reservoir.state = s_stop
  ...
  bgc.state = s_stop
  bgc.bgLvl = 125
  bgc.bgLvl_prev = 130
  ...
  STATE = s_stopping
```

Conclusion and Future Works

- Detailed Translation Mapping
- Mechanical application on multiple case studies
- Works in coordination with certain modelling restrictions –
  - Message payload only as integer type and restricted to a max count of 1
  - Incoming payload to be stored locally before use
  - An attribute is modified only once between 2 stable states
- Timer related issues for Multithreaded system designs, to be improvised
- Handling of enumerations and more as payload
- Reverse Translation from nuXmv counter traces, for model correction

UCA Universidad de Cádiz

FRANKFURT UNIVERSITY OF APPLIED SCIENCES

References

- Beaton, W.: Eclipse Papyrus for Real Time (Papyrus-RT). projects.eclipse.org (July 2017), [online]Available: https://www.eclipse.org/papyrus-rt/
- Rivet, C., Posse, E., Toolan, D.: Getting Started with Papyrus for Real Time v1.0. Survey of Requirements Management Standards - Eclipsepedia (Sept 2017), [online]Available: https://wiki.eclipse.org/Papyrus-RT/User/User_Guide/Getting_Started
- Hili, N., Posse, E., Dingel, U., Beaulieu, A.: Supporting Material For Eclipsecon'17 Unconference { Modeling & Analysis In Software Engineering (2017), [online]Available: https://flux.cs.queensu.ca/mase/papyrus-rt-resources/supporting-material-for-eclipsecon17-unconference/
- Sommerville, I.: An insulin pump control system. Software Engineering 10th Edition (Dec 2014)
- Bozzano, M., Cavada, R., Cimatti, A., Dorigatti, M., Griggio, A., Mariotti, A.,Micheli, A., Mover, S., Roveri, M., Tonetta, S.: nuXmv 1.1. 1 User Manual. FBK-Via Sommarive 18, 38055 (2016)

Thank you for your time !!