Translating a VDM Model to Kapture

Joe Hare, Leo Freitas and Ken Pierce

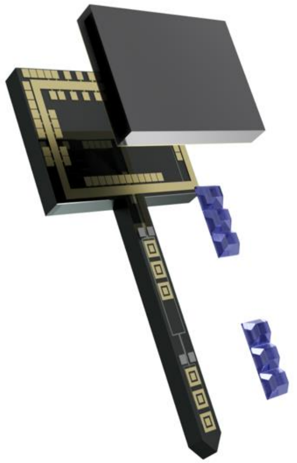
Overture 2025, 11/06/2025, Aarhus Denmark

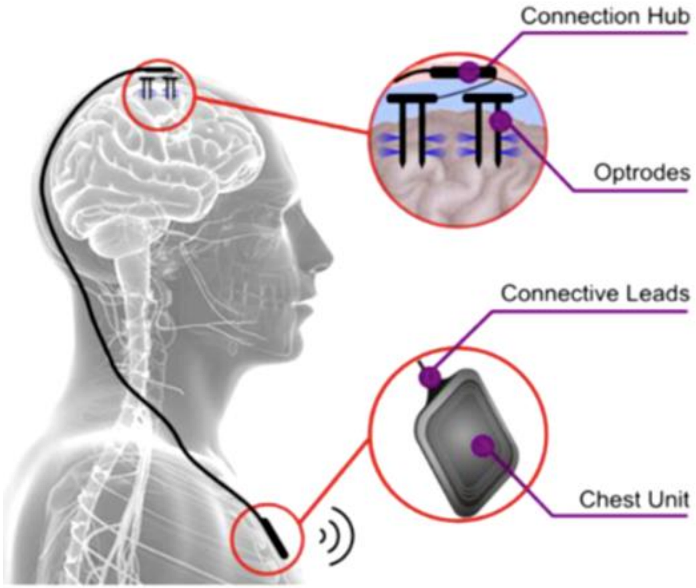
Motivation

- Explore the usefulness of a requirements modelling tool.
    - Assurance/safety case focused with links to DO178-C / DO-333 certification
    - Learn Kapture and begin modelling with it
        - Task undertaken by an undergraduate student with no prior FM experience
- Claim extra safety standards with Kapture
- Compare abstractions: Kapture x VDM-SL

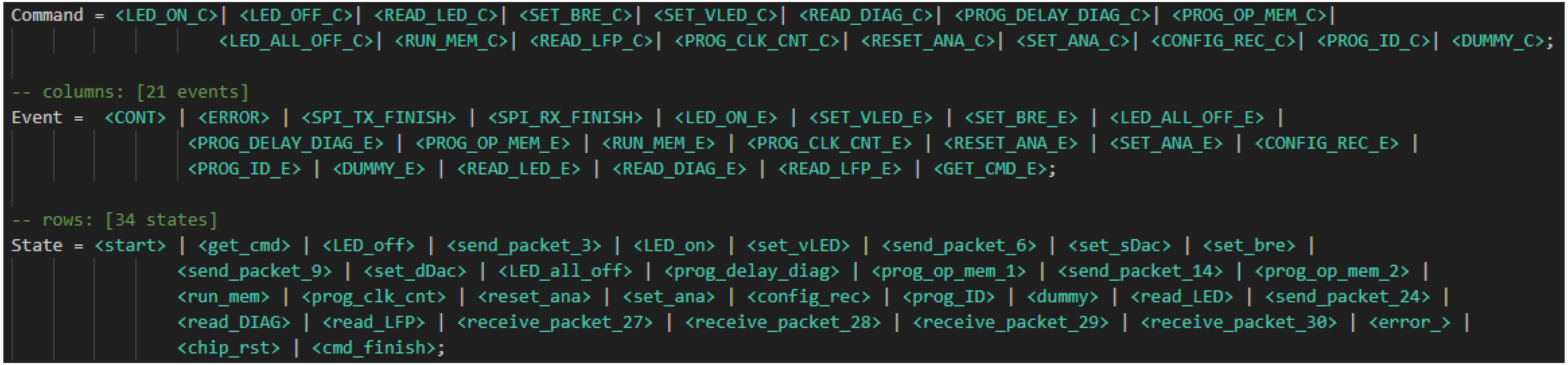
The Existing VDM model

- VDM-SL (1.5KLOC) abstraction of CANDO C code (3KLOC).
- Based on the CANDO optrode CMOS system for brain pace maker
- Model of the FSM controlling the behaviour of optrodes
  - State transitions are determined by the current state and an event variable
- Defines the operations run in each state



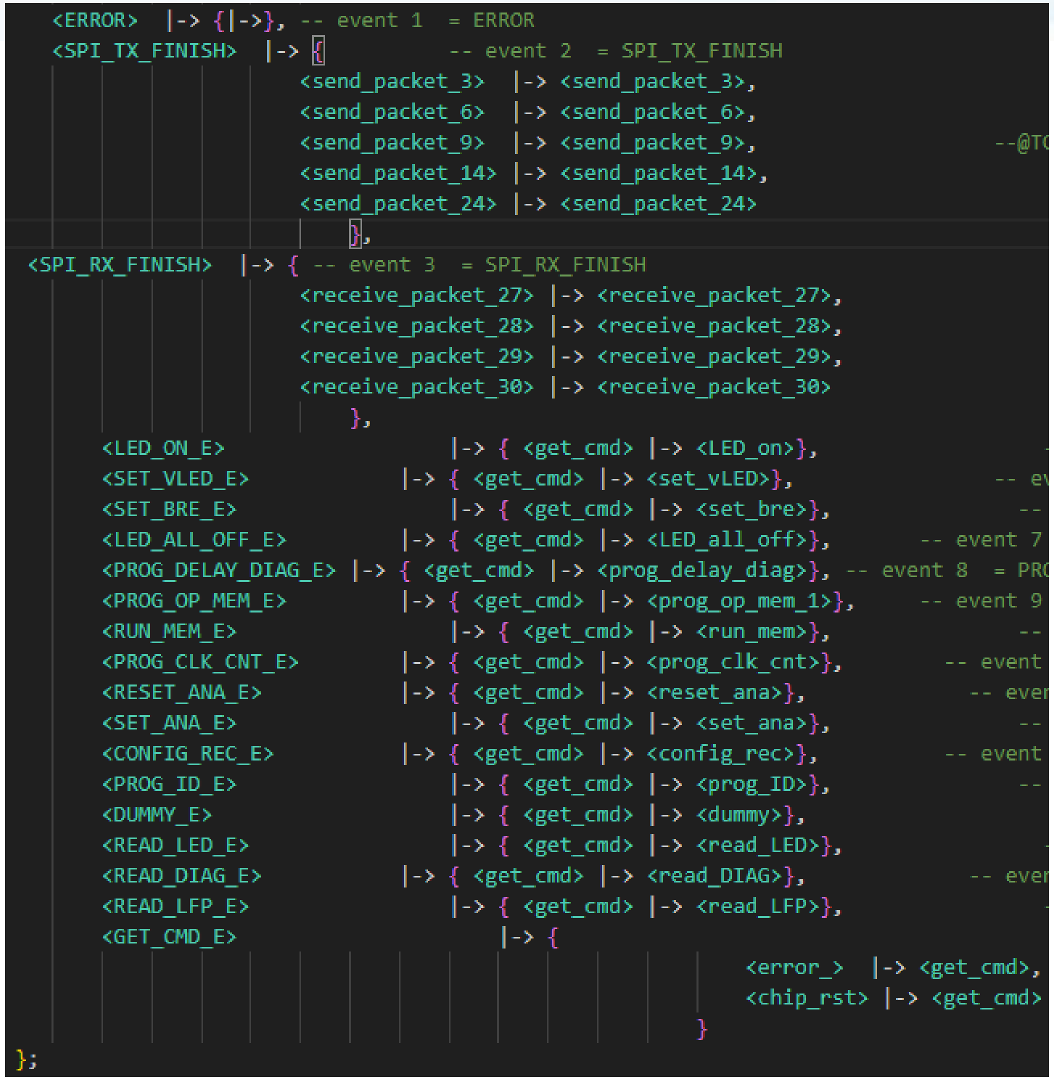


The Existing VDM model

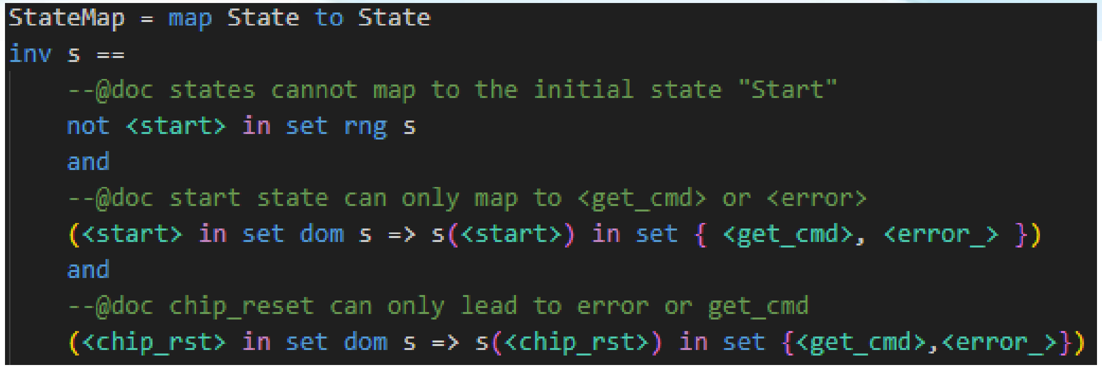


```
Command = <LED_ON_C>| <LED_OFF_C>| <READ_LED_C>| <SET_BRE_C>| <SET_VLED_C>| <READ_DIAG_C>| <PROG_DELAY_DIAG_C>| <PROG_OP_MEM_C>|
                    <LED_ALL_OFF_C>| <RUN_MEM_C>| <READ_LFP_C>| <PROG_CLK_CNT_C>| <RESET_ANA_C>| <SET_ANA_C>| <CONFIG_REC_C>| <PROG_ID_C>| <DUMMY_C>;

-- columns: [21 events]
Event =  <CONT> | <ERROR> | <SPI_TX_FINISH> | <SPI_RX_FINISH> | <LED_ON_E> | <SET_VLED_E> | <SET_BRE_E> | <LED_ALL_OFF_E> |
                 <PROG_DELAY_DIAG_E> | <PROG_OP_MEM_E> | <RUN_MEM_E> | <PROG_CLK_CNT_E> | <RESET_ANA_E> | <SET_ANA_E> | <CONFIG_REC_E> |
                 <PROG_ID_E> | <DUMMY_E> | <READ_LED_E> | <READ_DIAG_E> | <READ_LFP_E> | <GET_CMD_E>;

-- rows: [34 states]
State = <start> | <get_cmd> | <LED_off> | <send_packet_3> | <LED_on> | <set_vLED> | <send_packet_6> | <set_sDac> | <set_bre> |
                <send_packet_9> | <set_dDac> | <LED_all_off> | <prog_delay_diag> | <prog_op_mem_1> | <send_packet_14> | <prog_op_mem_2> |
                <run_mem> | <prog_clk_cnt> | <reset_ana> | <set_ana> | <config_rec> | <prog_ID> | <dummy> | <read_LED> | <send_packet_24> |
                <read_DIAG> | <read_LFP> | <receive_packet_27> | <receive_packet_28> | <receive_packet_29> | <receive_packet_30> | <error_> |
                <chip_rst> | <cmd_finish>;
```

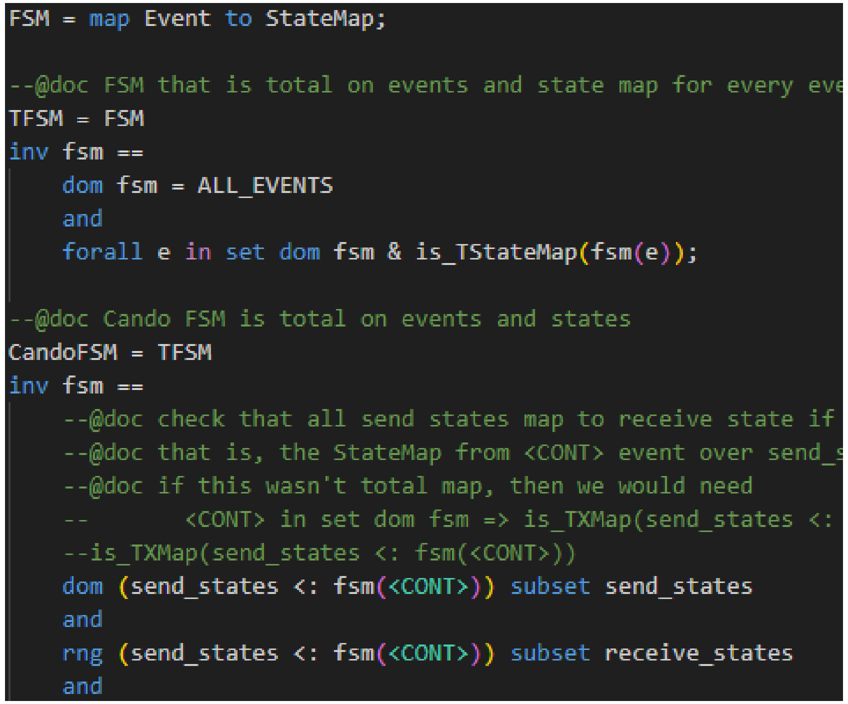
The Existing VDM Model



```
  <ERROR>  |-> {|->}, -- event 1  = ERROR
  <SPI_TX_FINISH>  |-> {           -- event 2  = SPI_TX_FINISH
                    <send_packet_3>  |-> <send_packet_3>,
                    <send_packet_6>  |-> <send_packet_6>,
                    <send_packet_9>  |-> <send_packet_9>,                    --@T(
                    <send_packet_14> |-> <send_packet_14>,
                    <send_packet_24> |-> <send_packet_24>
                        },
 <SPI_RX_FINISH>  |-> { -- event 3  = SPI_RX_FINISH
                    <receive_packet_27> |-> <receive_packet_27>,
                    <receive_packet_28> |-> <receive_packet_28>,
                    <receive_packet_29> |-> <receive_packet_29>,
                    <receive_packet_30> |-> <receive_packet_30>
                        },
      <LED_ON_E>                  |-> { <get_cmd> |-> <LED_on>},              -
      <SET_VLED_E>            |-> { <get_cmd> |-> <set_vLED>},               -- ev
      <SET_BRE_E>                 |-> { <get_cmd> |-> <set_bre>},              --
      <LED_ALL_OFF_E>         |-> { <get_cmd> |-> <LED_all_off>},       -- event 7
      <PROG_DELAY_DIAG_E> |-> { <get_cmd> |-> <prog_delay_diag>}, -- event 8  = PRO
      <PROG_OP_MEM_E>         |-> { <get_cmd> |-> <prog_op_mem_1>},     -- event 9
      <RUN_MEM_E>                 |-> { <get_cmd> |-> <run_mem>},              --
      <PROG_CLK_CNT_E>        |-> { <get_cmd> |-> <prog_clk_cnt>},        -- event
      <RESET_ANA_E>           |-> { <get_cmd> |-> <reset_ana>},             -- even
      <SET_ANA_E>                 |-> { <get_cmd> |-> <set_ana>},              --
      <CONFIG_REC_E>          |-> { <get_cmd> |-> <config_rec>},          -- event
      <PROG_ID_E>                 |-> { <get_cmd> |-> <prog_ID>},              --
      <DUMMY_E>                   |-> { <get_cmd> |-> <dummy>},
      <READ_LED_E>                |-> { <get_cmd> |-> <read_LED>},             -
      <READ_DIAG_E>           |-> { <get_cmd> |-> <read_DIAG>},             -- even
      <READ_LFP_E>                |-> { <get_cmd> |-> <read_LFP>},             -
      <GET_CMD_E>                     |-> {
                                              <error_>  |-> <get_cmd>,
                                              <chip_rst> |-> <get_cmd>
                                          }
};
```

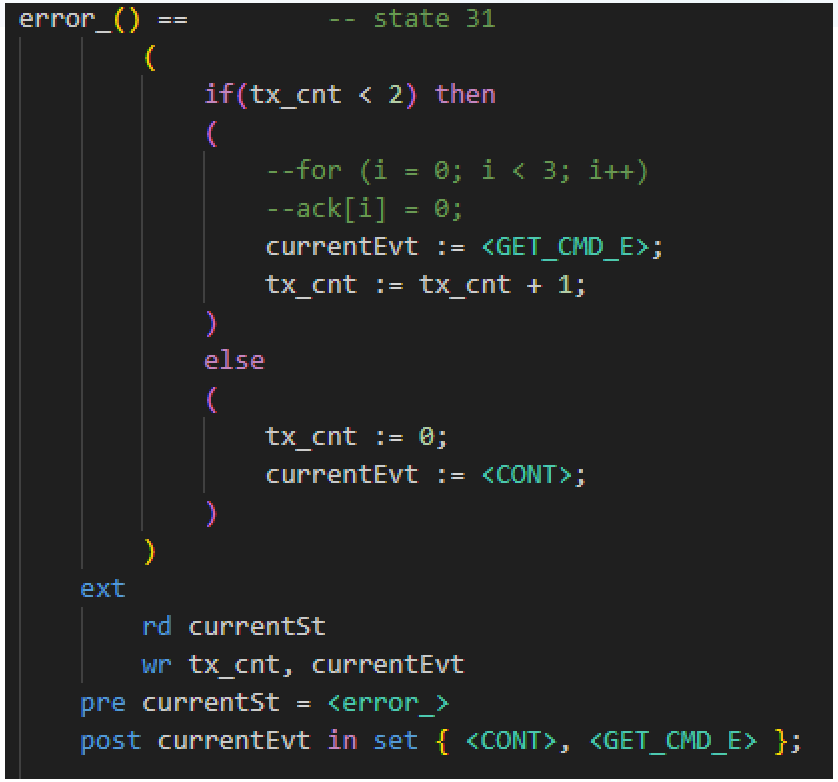


```
StateMap = map State to State
inv s ==
    --@doc states cannot map to the initial state "Start"
    not <start> in set rng s
    and
    --@doc start state can only map to <get_cmd> or <error>
    (<start> in set dom s => s(<start>) in set { <get_cmd>, <error_> })
    and
    --@doc chip_reset can only lead to error or get_cmd
    (<chip_rst> in set dom s => s(<chip_rst>) in set {<get_cmd>,<error_>})
```

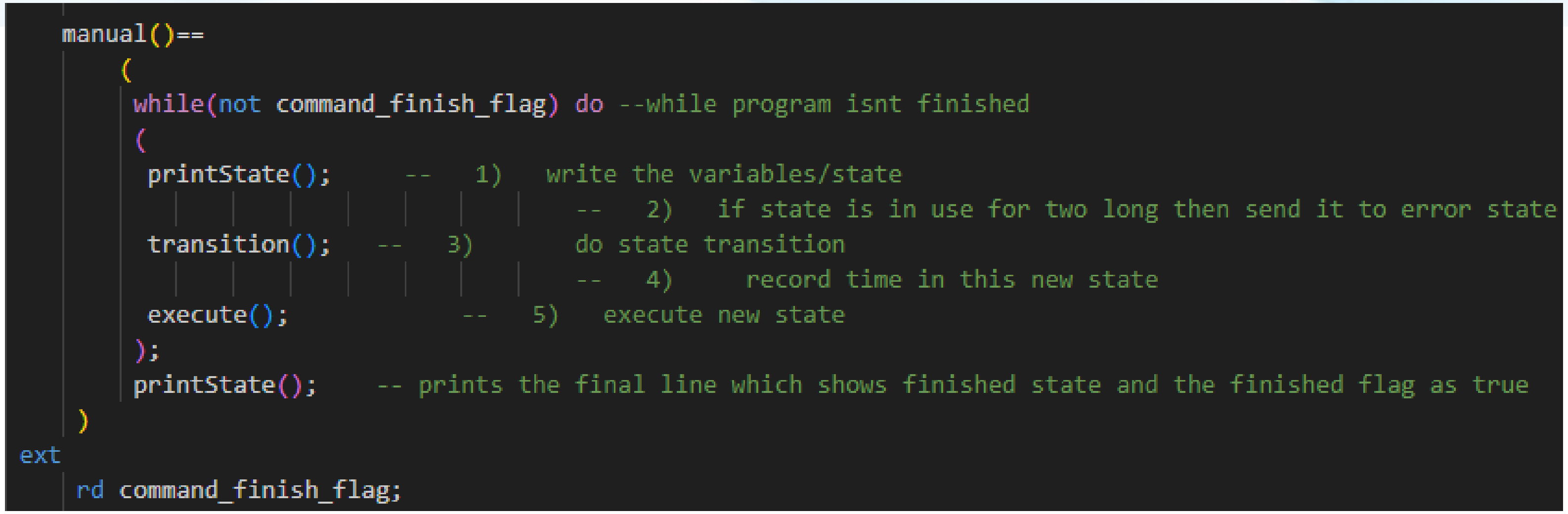


```
FSM = map Event to StateMap;

--@doc FSM that is total on events and state map for every eve
TFSM = FSM
inv fsm ==
    dom fsm = ALL_EVENTS
    and
    forall e in set dom fsm & is_TStateMap(fsm(e));

--@doc Cando FSM is total on events and states
CandoFSM = TFSM
inv fsm ==
    --@doc check that all send states map to receive state if
    --@doc that is, the StateMap from <CONT> event over send_s
    --@doc if this wasn't total map, then we would need
    --        <CONT> in set dom fsm => is_TXMap(send_states <:
    --is_TXMap(send_states <: fsm(<CONT>))
    dom (send_states <: fsm(<CONT>)) subset send_states
    and
    rng (send_states <: fsm(<CONT>)) subset receive_states
    and
```

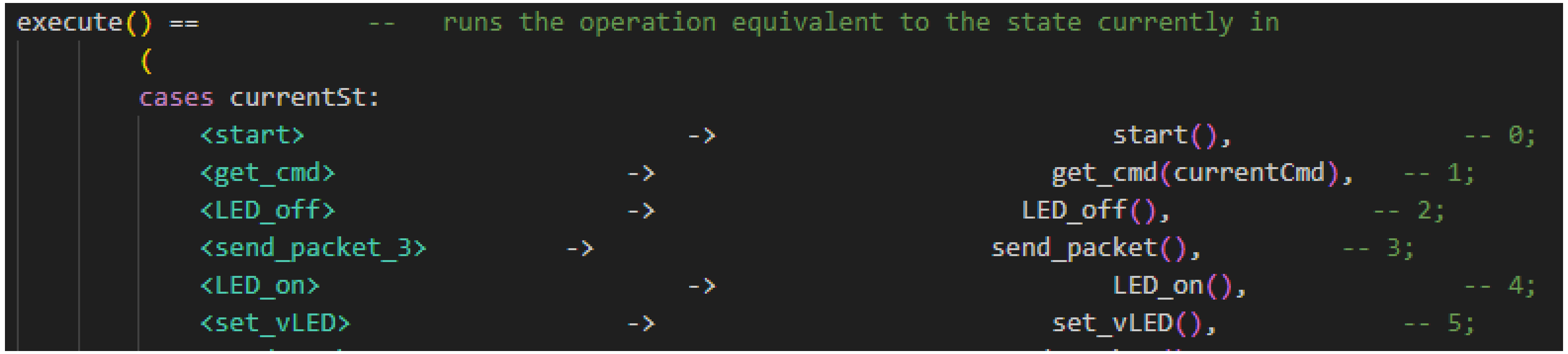
The Existing VDM Model



```
error_() ==          -- state 31
        (
            if(tx_cnt < 2) then
            (
                --for (i = 0; i < 3; i++)
                --ack[i] = 0;
                currentEvt := <GET_CMD_E>;
                tx_cnt := tx_cnt + 1;
            )
            else
            (
                tx_cnt := 0;
                currentEvt := <CONT>;
            )
        )
    ext
        rd currentSt
        wr tx_cnt, currentEvt
    pre currentSt = <error_>
    post currentEvt in set { <CONT>, <GET_CMD_E> };
```



```
   manual()==
       (
        while(not command_finish_flag) do --while program isnt finished
        (
         printState();     --   1)   write the variables/state
                                 --   2)   if state is in use for two long then send it to error state
         transition();   --   3)       do state transition
                                 --   4)      record time in this new state
         execute();              --   5)   execute new state
        );
        printState();    -- prints the final line which shows finished state and the finished flag as true
    )
ext
    rd command_finish_flag;
```

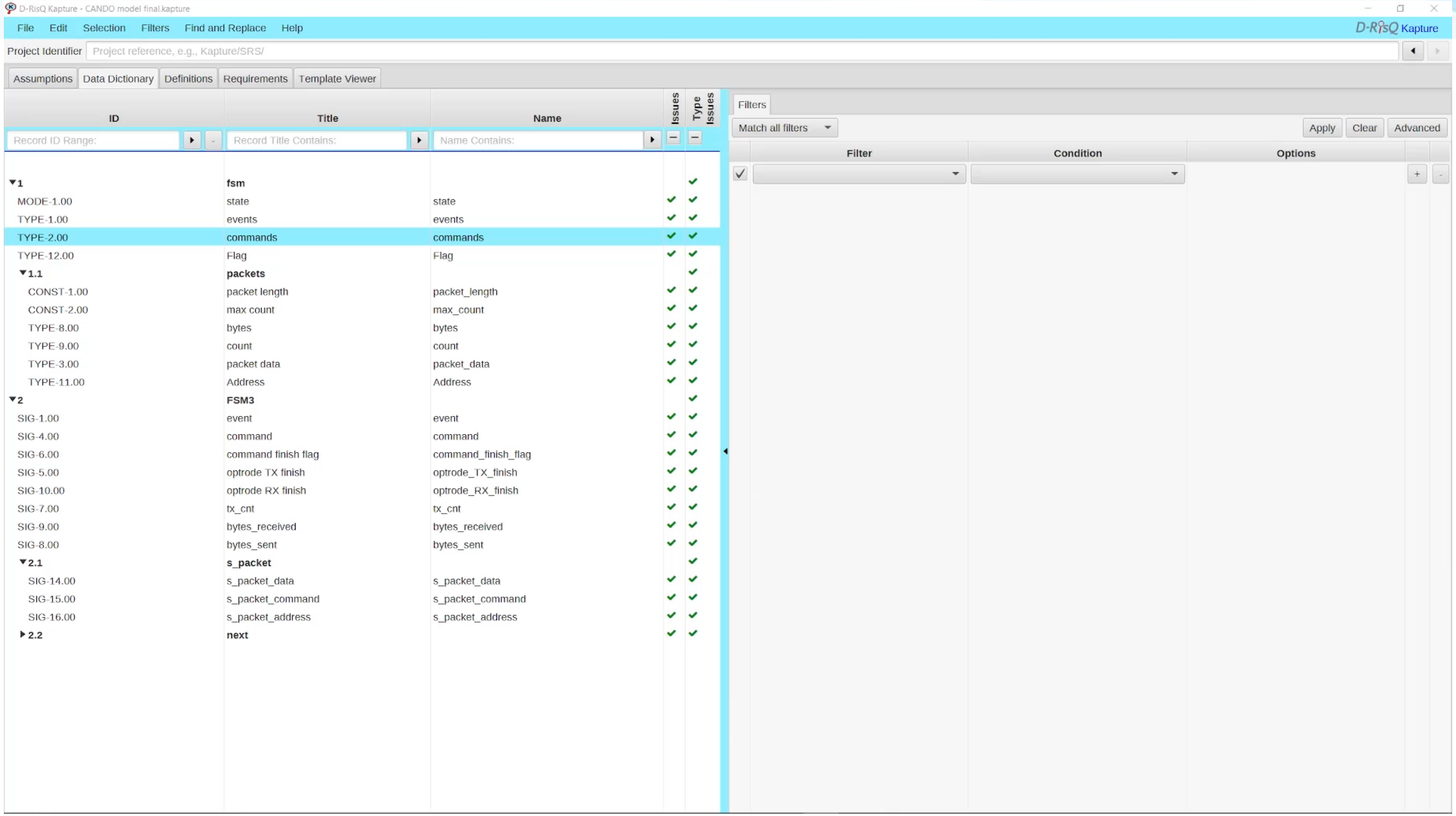


```
execute() ==            --   runs the operation equivalent to the state currently in
        (
        cases currentSt:
            <start>                   ->                        start(),           -- 0;
            <get_cmd>               ->                        get_cmd(currentCmd),   -- 1;
            <LED_off>               ->                        LED_off(),           -- 2;
            <send_packet_3>       ->                        send_packet(),       -- 3;
            <LED_on>                  ->                        LED_on(),            -- 4;
            <set_vLED>              ->                        set_vLED(),          -- 5;
```

An Overview of Kapture

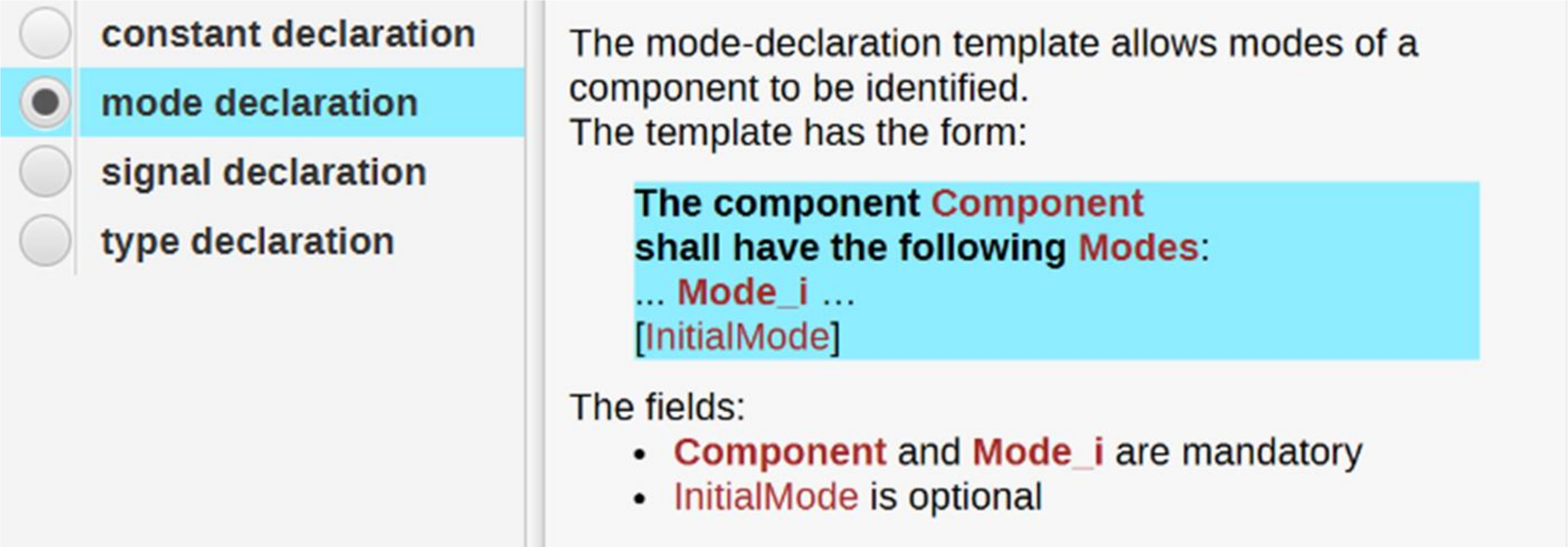
- A tool for writing clear software requirements in English
- English constructs Converted to CSP/Stateflow for validation
- Final conversion to Matlab Simulink Stateflow
- Designed for high assurance
- calculating C code from high-level requirements (ClawZ)
- Compliance with DO-178C and DO-333
- Applied to various industries (nuclear, avionics, medical)
- Part of an EU innovate grant with industry partners

An Overview of Kapture



ID	Title	Name	Issues	Type Issues
▼1	**fsm**			✓
MODE-1.00	state	state	✓	✓
TYPE-1.00	events	events	✓	✓
TYPE-2.00	commands	commands	✓	✓
TYPE-12.00	Flag	Flag	✓	✓
▼1.1	**packets**			✓
CONST-1.00	packet length	packet_length	✓	✓
CONST-2.00	max count	max_count	✓	✓
TYPE-8.00	bytes	bytes	✓	✓
TYPE-9.00	count	count	✓	✓
TYPE-3.00	packet data	packet_data	✓	✓
TYPE-11.00	Address	Address	✓	✓
▼2	**FSM3**			✓
SIG-1.00	event	event	✓	✓
SIG-4.00	command	command	✓	✓
SIG-6.00	command finish flag	command_finish_flag	✓	✓
SIG-5.00	optrode TX finish	optrode_TX_finish	✓	✓
SIG-10.00	optrode RX finish	optrode_RX_finish	✓	✓
SIG-7.00	tx_cnt	tx_cnt	✓	✓
SIG-9.00	bytes_received	bytes_received	✓	✓
SIG-8.00	bytes_sent	bytes_sent	✓	✓
▼2.1	**s_packet**			✓
SIG-14.00	s_packet_data	s_packet_data	✓	✓
SIG-15.00	s_packet_command	s_packet_command	✓	✓
SIG-16.00	s_packet_address	s_packet_address	✓	✓
▶2.2	**next**		✓	✓

Model Translation: States and Events



○ **constant declaration**

● **mode declaration**

○ **signal declaration**

○ **type declaration**

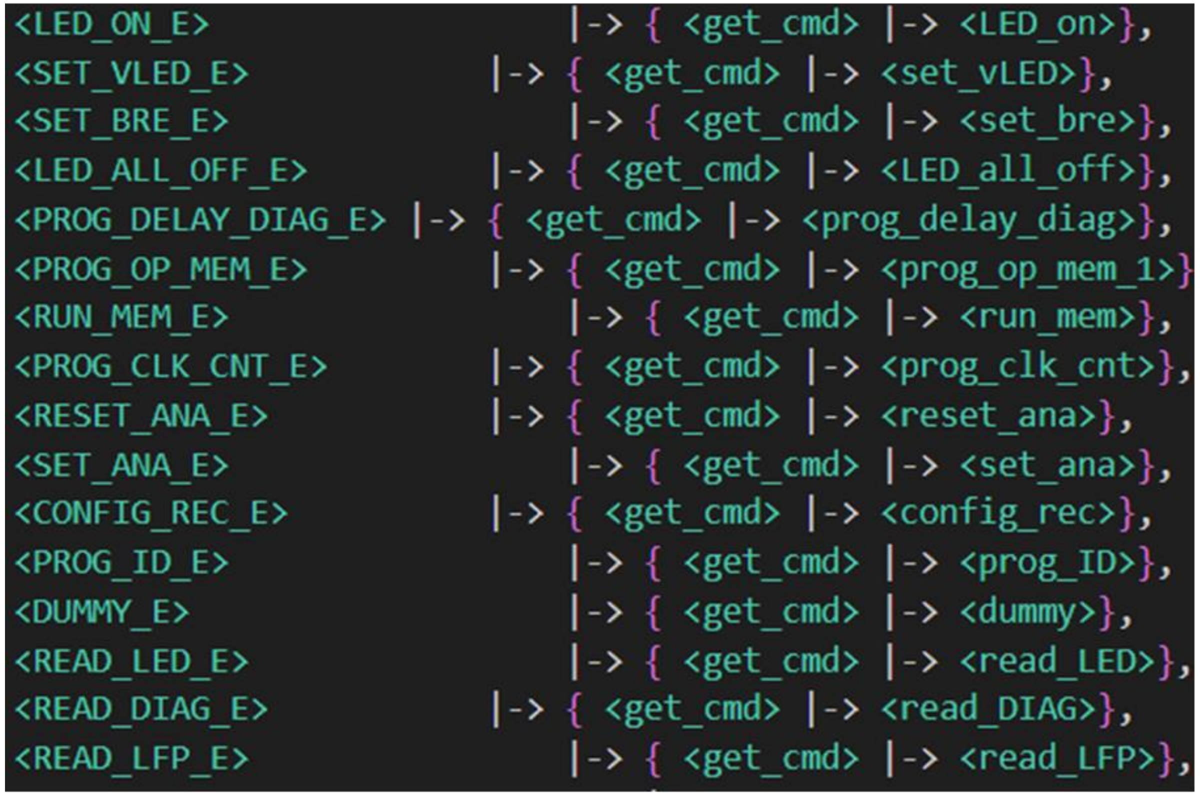
The mode-declaration template allows modes of a component to be identified.
The template has the form:

> **The component Component**
> **shall have the following Modes:**
> ... **Mode_i** ...
> [InitialMode]

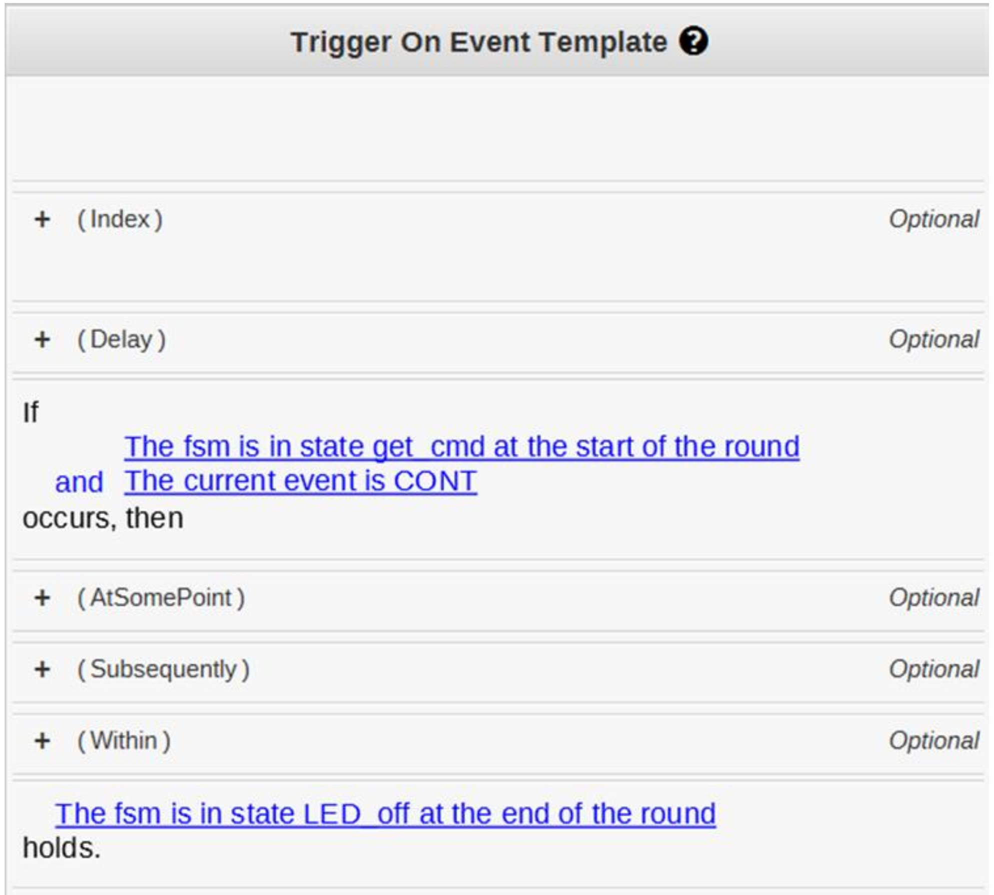
The fields:

- **Component** and **Mode_i** are mandatory
- InitialMode is optional

Model Translation: Rounds



```
<LED_ON_E>                    |-> { <get_cmd> |-> <LED_on>},
<SET_VLED_E>              |-> { <get_cmd> |-> <set_vLED>},
<SET_BRE_E>                   |-> { <get_cmd> |-> <set_bre>},
<LED_ALL_OFF_E>           |-> { <get_cmd> |-> <LED_all_off>},
<PROG_DELAY_DIAG_E> |-> { <get_cmd> |-> <prog_delay_diag>},
<PROG_OP_MEM_E>           |-> { <get_cmd> |-> <prog_op_mem_1>}
<RUN_MEM_E>                   |-> { <get_cmd> |-> <run_mem>},
<PROG_CLK_CNT_E>          |-> { <get_cmd> |-> <prog_clk_cnt>},
<RESET_ANA_E>             |-> { <get_cmd> |-> <reset_ana>},
<SET_ANA_E>                   |-> { <get_cmd> |-> <set_ana>},
<CONFIG_REC_E>            |-> { <get_cmd> |-> <config_rec>},
<PROG_ID_E>                   |-> { <get_cmd> |-> <prog_ID>},
<DUMMY_E>                     |-> { <get_cmd> |-> <dummy>},
<READ_LED_E>                  |-> { <get_cmd> |-> <read_LED>},
<READ_DIAG_E>             |-> { <get_cmd> |-> <read_DIAG>},
<READ_LFP_E>                  |-> { <get_cmd> |-> <read_LFP>},
```



Trigger On Event Template

+ (Index) *Optional*

+ (Delay) *Optional*

If
The fsm is in state get_cmd at the start of the round
and The current event is CONT
occurs, then

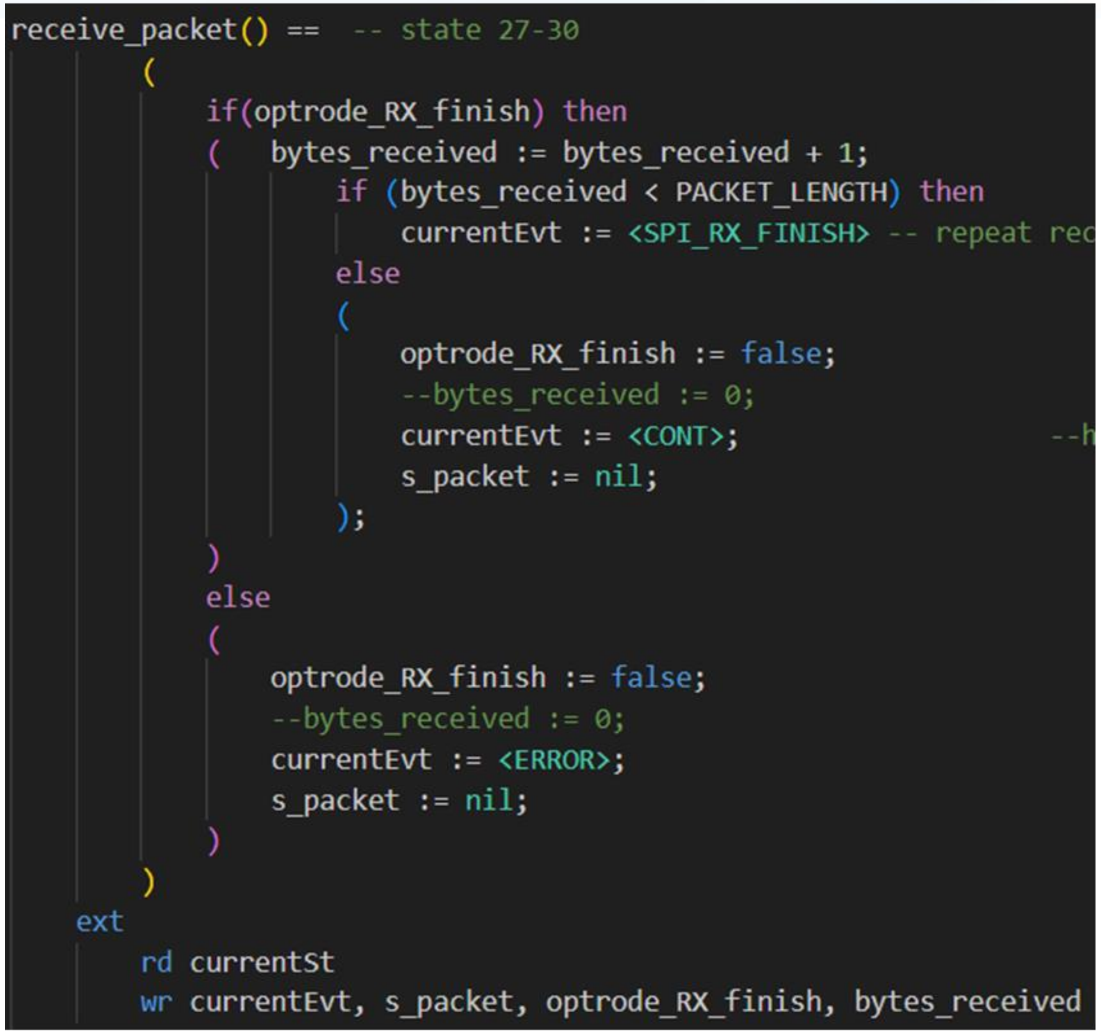
+ (AtSomePoint) *Optional*

+ (Subsequently) *Optional*

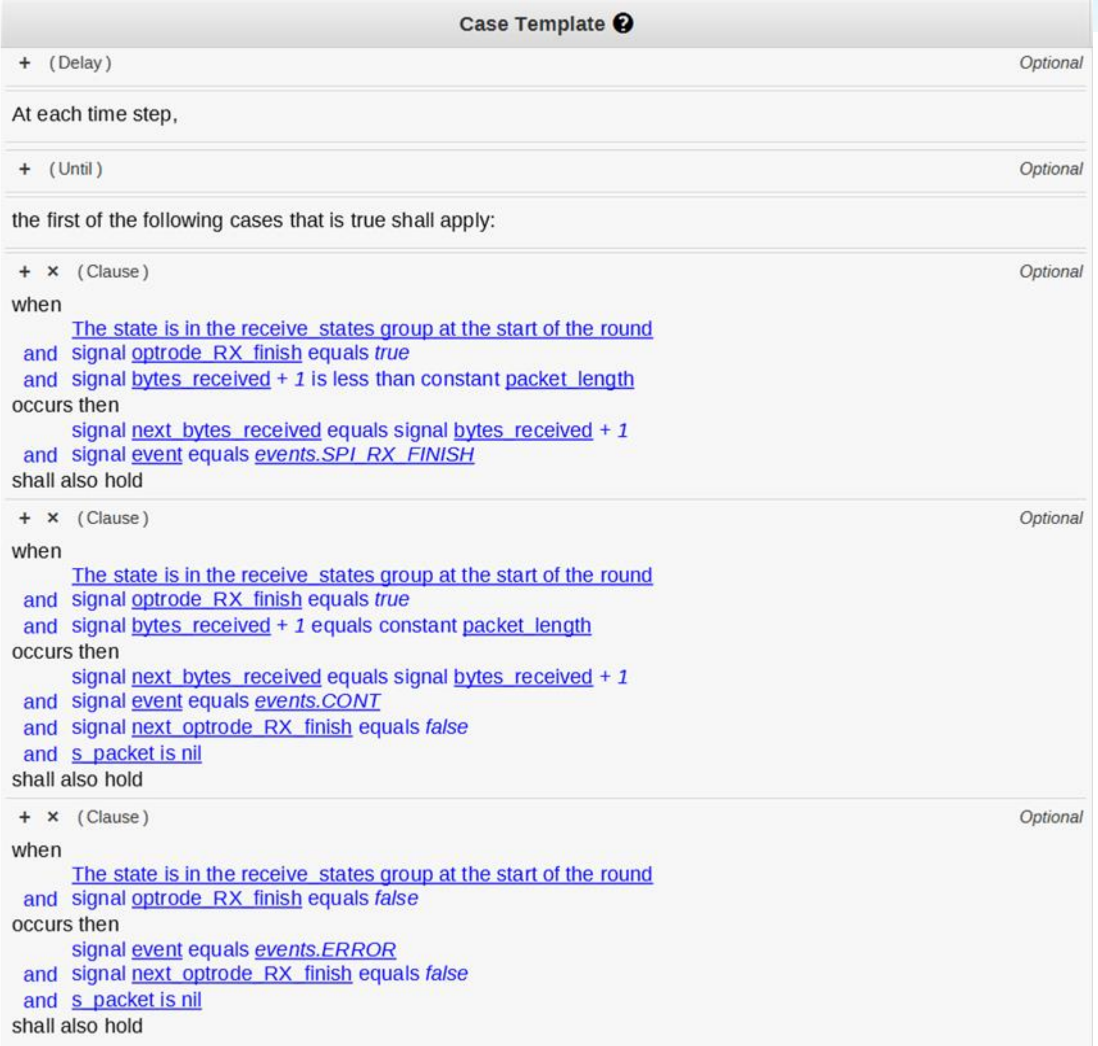
+ (Within) *Optional*

The fsm is in state LED_off at the end of the round
holds.

Model Translation: Operations



```
receive_packet() ==  -- state 27-30
        (
            if(optrode_RX_finish) then
            (   bytes_received := bytes_received + 1;
                if (bytes_received < PACKET_LENGTH) then
                    currentEvt := <SPI_RX_FINISH> -- repeat rec
                else
                (
                    optrode_RX_finish := false;
                    --bytes_received := 0;
                    currentEvt := <CONT>;                    --h
                    s_packet := nil;
                );
            )
            else
            (
                optrode_RX_finish := false;
                --bytes_received := 0;
                currentEvt := <ERROR>;
                s_packet := nil;
            )
        )
    ext
        rd currentSt
        wr currentEvt, s_packet, optrode_RX_finish, bytes_received
```



Case Template ?

+ (Delay) *Optional*

At each time step,

+ (Until) *Optional*

the first of the following cases that is true shall apply:

+ × (Clause) *Optional*

when
The state is in the receive_states group at the start of the round
and signal optrode_RX_finish equals *true*
and signal bytes_received + *1* is less than constant packet_length
occurs then
signal next_bytes_received equals signal bytes_received + *1*
and signal event equals *events.SPI_RX_FINISH*
shall also hold

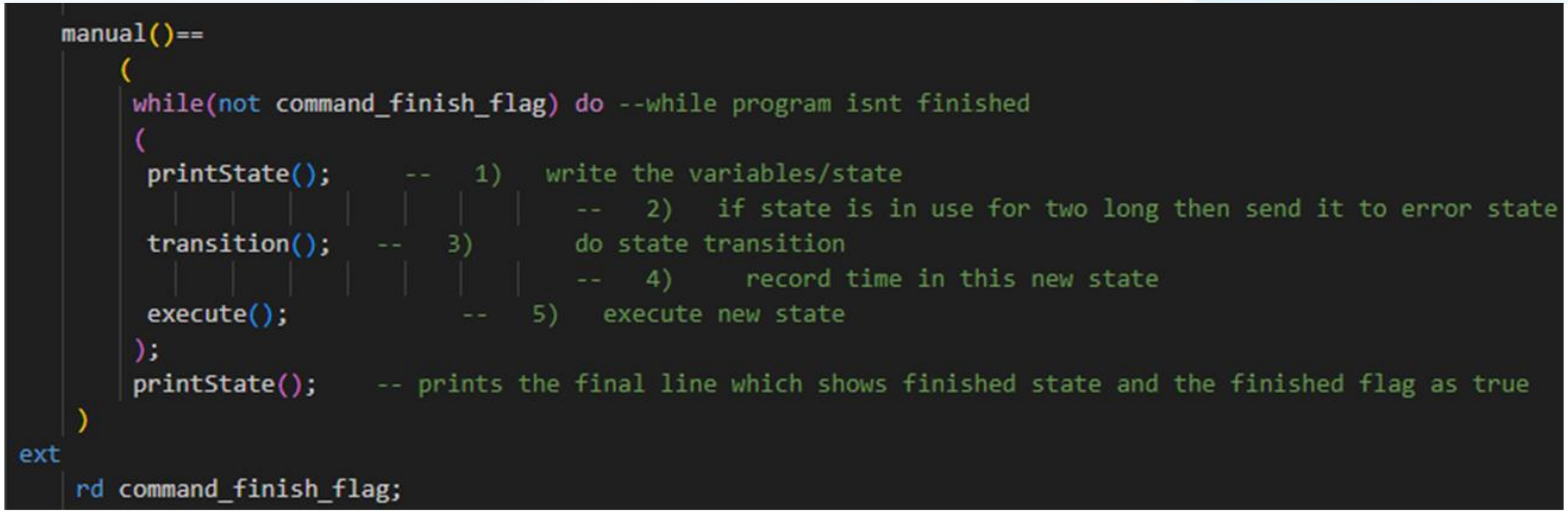
+ × (Clause) *Optional*

when
The state is in the receive_states group at the start of the round
and signal optrode_RX_finish equals *true*
and signal bytes_received + *1* equals constant packet_length
occurs then
signal next_bytes_received equals signal bytes_received + *1*
and signal event equals *events.CONT*
and signal next_optrode_RX_finish equals *false*
and s_packet is nil
shall also hold

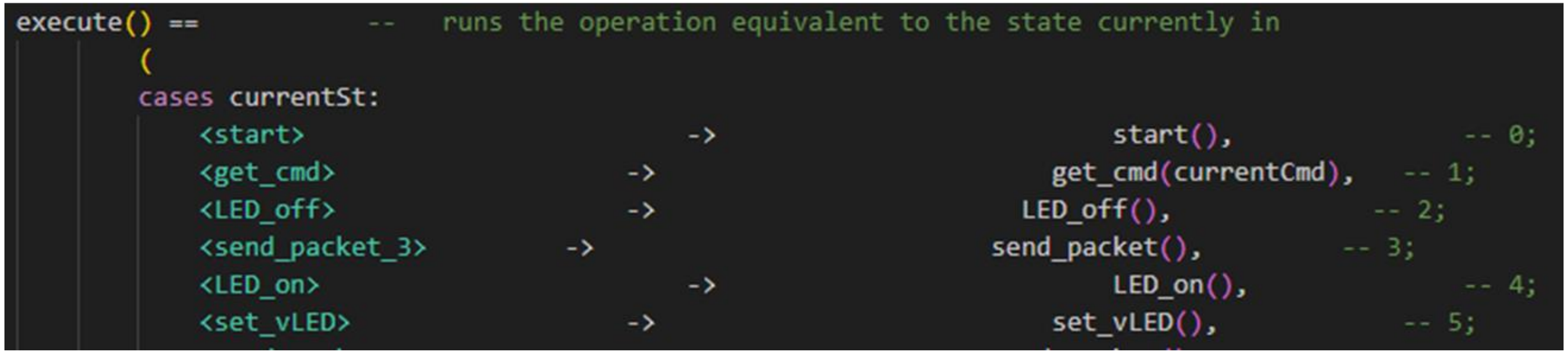
+ × (Clause) *Optional*

when
The state is in the receive_states group at the start of the round
and signal optrode_RX_finish equals *false*
occurs then
signal event equals *events.ERROR*
and signal next_optrode_RX_finish equals *false*
and s_packet is nil
shall also hold

Model Translation: Operations

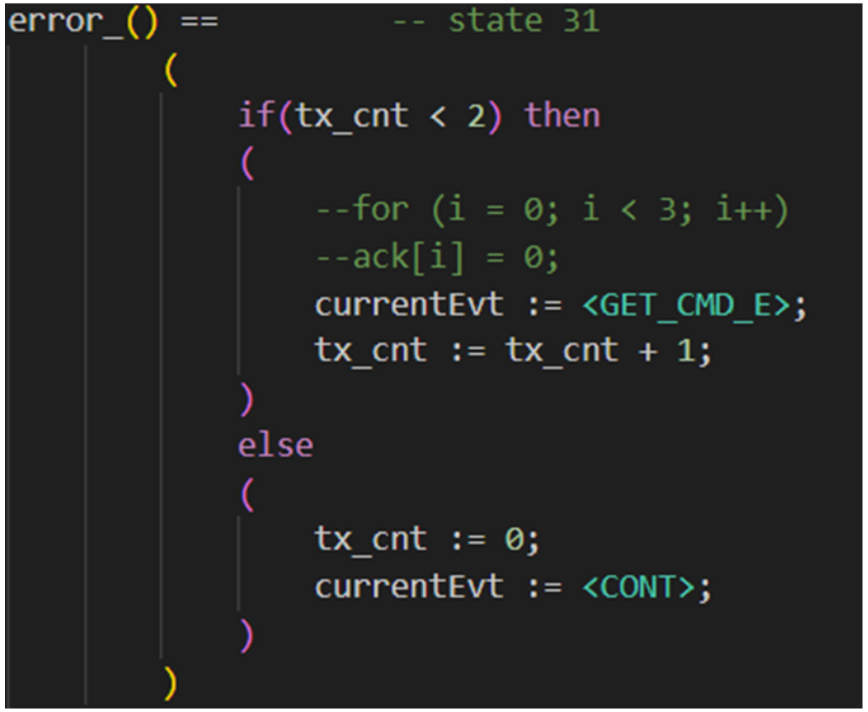


```
   manual()==
       (
        while(not command_finish_flag) do --while program isnt finished
        (
         printState();     --   1)   write the variables/state
                                      --   2)   if state is in use for two long then send it to error state
         transition();   --   3)       do state transition
                                      --   4)      record time in this new state
         execute();              --   5)   execute new state
        );
        printState();    -- prints the final line which shows finished state and the finished flag as true
     )
ext
    rd command_finish_flag;
```

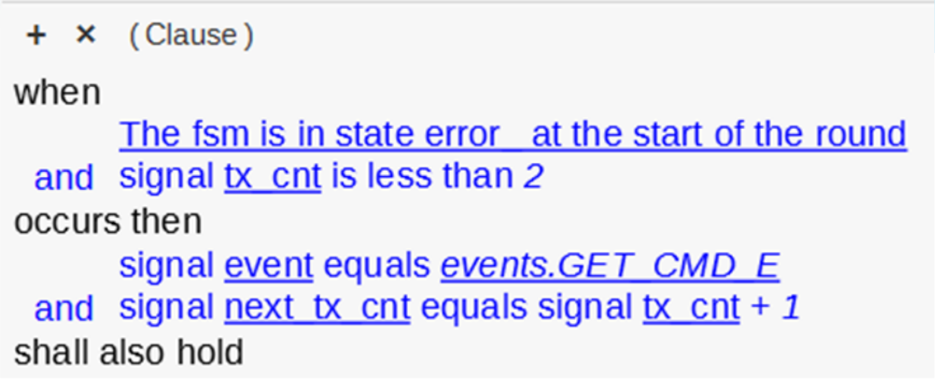


```
execute() ==            --   runs the operation equivalent to the state currently in
         (
         cases currentSt:
             <start>                    ->                          start(),              -- 0;
             <get_cmd>               ->                   get_cmd(currentCmd),    -- 1;
             <LED_off>               ->                  LED_off(),               -- 2;
             <send_packet_3>       ->                send_packet(),          -- 3;
             <LED_on>                  ->                      LED_on(),              -- 4;
             <set_vLED>              ->                   set_vLED(),            -- 5;
```

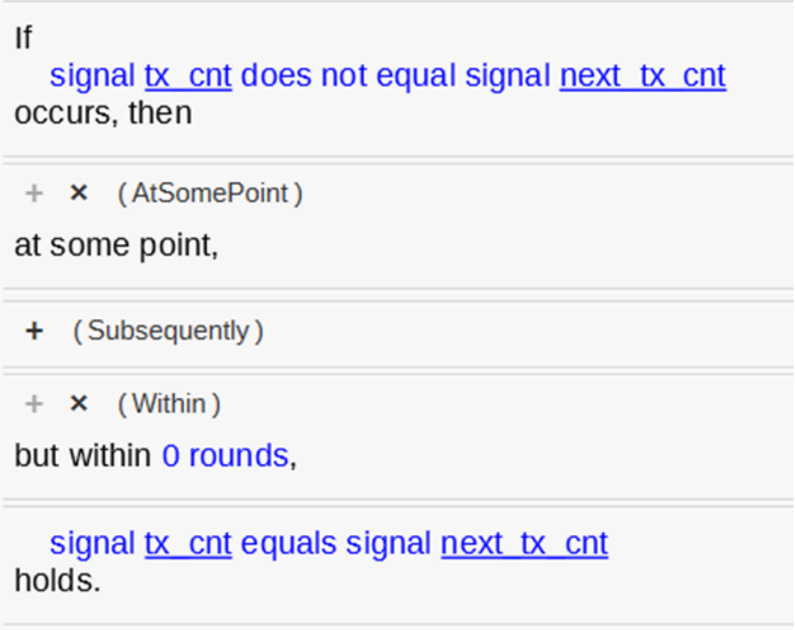
Incrementing values issue



```
error_() ==           -- state 31
    (
        if(tx_cnt < 2) then
        (
            --for (i = 0; i < 3; i++)
            --ack[i] = 0;
            currentEvt := <GET_CMD_E>;
            tx_cnt := tx_cnt + 1;
        )
        else
        (
            tx_cnt := 0;
            currentEvt := <CONT>;
        )
    )
```



when
    The fsm is in state error_ at the start of the round
 and signal tx_cnt is less than *2*
occurs then
    signal event equals *events.GET_CMD_E*
 and signal next_tx_cnt equals signal tx_cnt + *1*
shall also hold



If
  signal tx_cnt does not equal signal next_tx_cnt
occurs, then

at some point,

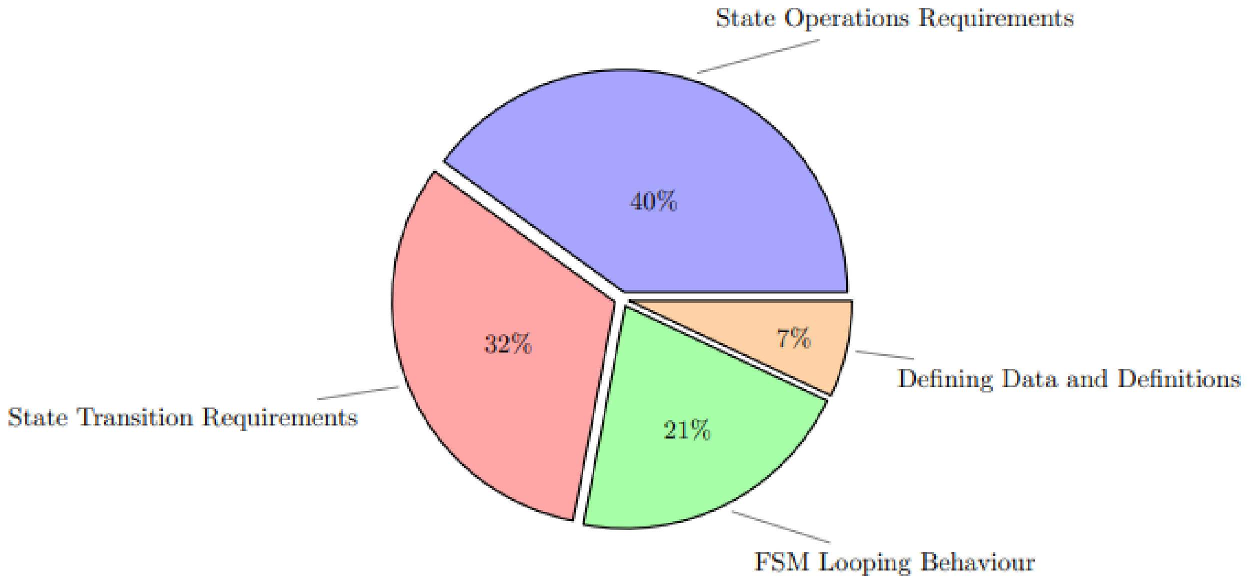
but within 0 rounds,

  signal tx_cnt equals signal next_tx_cnt
holds.

Results and Testing

- Results tested by creating manual traces using the Kapture model
  - Traces had same results as the ones generated by VDM-SL model
- Completed within four weeks
- >90% of the (~1000 LOC) VDM model translated into 113 Kapture requirements

Proportion of time spent translating different parts of VDM model into Kapture



Issues Encountered During Development

- Learning curve
- Lack of tools for creating and editing en masse
- Using Kapture for low-level requirements
- Shift in abstraction between VDM and Kapture
  - Feedback from D-RisQ

Conclusions and Future Work

- Successful translation in a short period of time
- A readable set of requirements for CANDO

- Further improvements to the model
- Modelworks
- Further collaboration with D-RisQ