Towards the integration of Overture and TASTE

T. Fabbri [1], M. Verhoef [2], V. Bandur [3], M. Perrotin [2], T. Tsiodras [2], P.G. Larsen [3]
with many thanks to K.G. Lausdahl and P.W.V. Tran-Jørgensen

work performed as a ESA Summer Of Code in Space project by T. Fabbri

[1] University of Pisa, department of Information Engineering (I)
[2] European Space Agency, ESTEC (NL)
[3] Aarhus University, department of engineering (DK)

Agenda of this talk

- Recap from Overture-13
  - Introduction to TASTE
  - Why do we want to integrate Overture into TASTE?
- Experiment #1 : Model-level integration of Overture and TASTE
- Experiment #2 : Code-level integration of Overture and TASTE
- Conclusions and future work

The TASTE toolset (1)

Consolidated result from (and continued development of) the ESA-led EU-FP6 ASSERT project: **T**he **A**SSERT **S**et of **T**ools for **E**ngineering:

- open-source tool suite for rigorous software engineering
- aimed at development of heterogeneous embedded systems
- focus on (but not limited to) space on-board software (reliability, qualification)
- based on mature (formal) notations with long term support
- model-centric development with high levels of automation (suited for agile)
- seamless interoperability offers DSL-like approach
- model synthesis towards wide range of target platforms
- robust tools maintained by active (but small) community

Increase *developer productivity* by providing for *automated system synthesis* and *continuous integration* by exploiting well-founded *rigorous modeling techniques*

For more information see http://taste.tools/

The TASTE toolset (2)

The main elements of TASTE are:

1. Abstract Syntax Notation One (ASN.1, ITU X.680-X.693)

- used to describe (abstract) data types (i.e. TC and TM)
- orthogonal encoding rules for physical representation
- (qualified) code generation and run-time support for C and Ada
- generation of interface documentation and test sets

2. Architecture Analysis and Design Language (AADL, SAE AS 5506B)

- extensible formal textual and graphical notation
- used to describe the system logical and physical architecture
- used to capture avionics hardware components,
  their communication interfaces and deployment of software artifacts
- generation of high-integrity (SPARK) Ada code

The TASTE toolset (3)

The main elements of TASTE are (continued):

3. Specification and Description Language (SDL, ITU-T Rec. Z.100)

- formal language to describe *state machines*
- graphical and textual notation, native support for ASN.1 types
- model evolution visualized as *message sequence chart* (Z.150)
- record and playback useful for analysis and testing
- code generation to (SPARK) Ada

4. Build automation and automatic target deployment

- Light-weight, portable and qualifiable run-time: PolyORB Hi-C / Hi-Ada
- Linux and SMP2 simulation environments
- RTEMS and Xenomai on (virtualized) QEMU or TSIM
- RTEMS or Ada-Ravenscar run-time on target hardware

(caveat: also support for Simulink, SCADE, VHDL, Ada, C)

The TASTE toolset (4)

The TASTE development process consist of the following steps:

1. describe the system logical architecture (AADL) and interfaces (ASN.1)
2. describe the system behavior (SDL, VHDL, C, Ada, Scade, Simulink)
3. describe the deployment of functionality on the avionics (AADL)
4. generate code, build the system and download on simulator or target
5. monitor and interact with the system at run-time (test execution)
6. iterate

TASTE allows complementary analysis (re-)using the constituent models

- Enhanced verification and validation (testing, model checking)
- Schedulability analysis using MAST and CHEDDAR tools on AADL models
- Use AADL extension capability to specify explicit fault behavior using *System-Level Integrated Modelling* language (SLIM) which can be verified using the (TASTE compatible) COMPASS tools (nuSMV)

(caveat: complementary analysis is possible <u>before</u> system behavior is complete)

Comparing TASTE to Overture

TASTE	OVERTURE
open-source	open-source
robust tool set (quality control)	robust tool set (quality control)
research platform (explore new ideas)	research platform (explore new ideas)
(co-)simulation to support validation	(co-)simulation to support validation
focus on rigorous analysis and testing	focus on rigorous analysis and testing
small but active community	small but active community

COMMONALITIES (STRENGTHS)

TASTE	OVERTURE
improving quality of code artifacts	early design validation
goal is to extend scope towards modeling	goal is to extend scope towards synthesis
built-in support for state machines	state machines require framework
weak support for data transformations	strong support for data transformations

COMPLEMENTARY (OPPORTUNITIES)

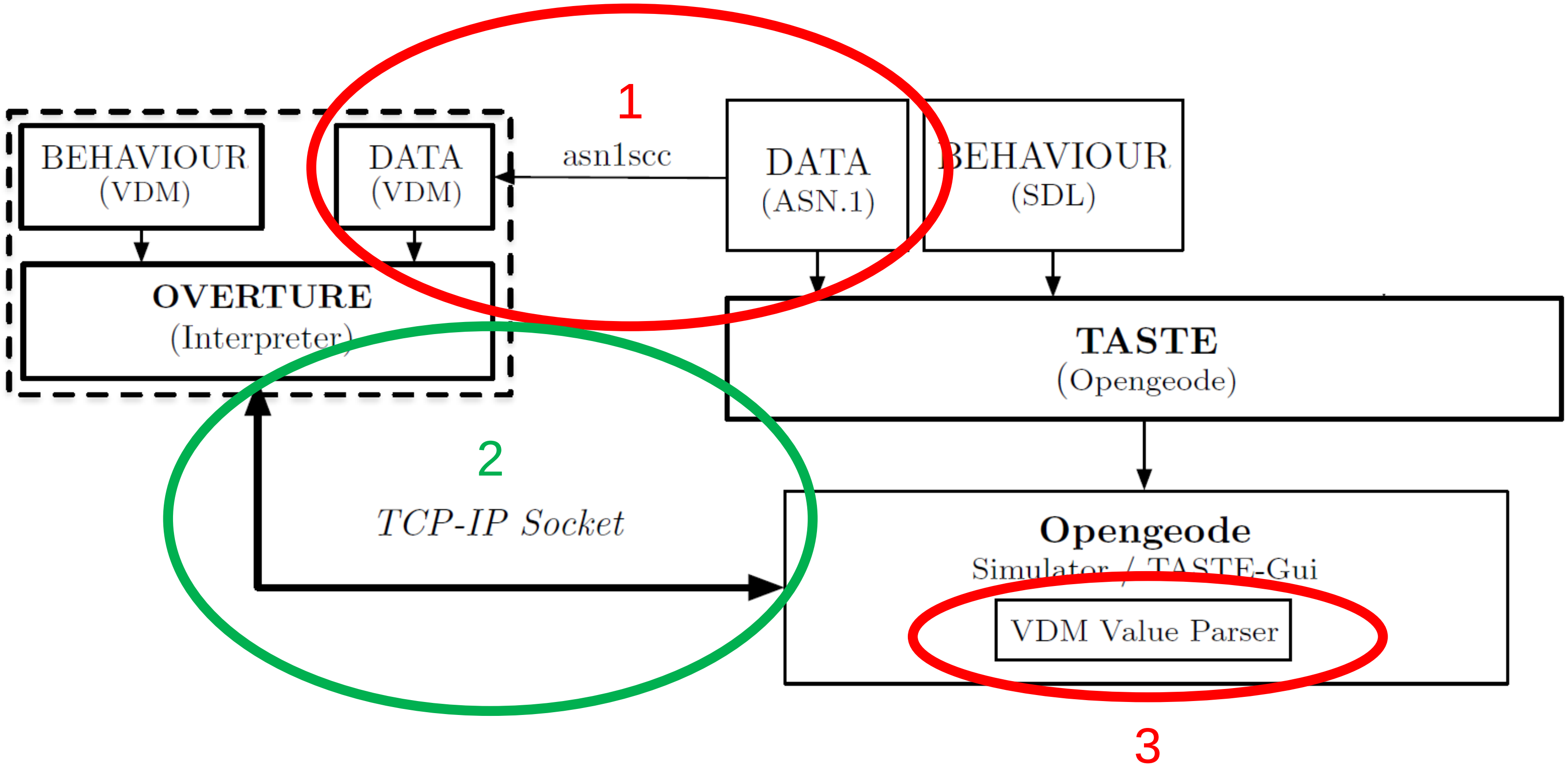
TASTE	OVERTURE
extensible architecture model	restricted built-in architecture model
implemented in python / QT on Linux	Eclipse based (multi platform)

DIFFERENCES (WEAKNESS)

Integrate Overture as a first-class citizen into TASTE

1. couple the Overture interpreter to OpenGEODE (simulation, exploration)
2. Integrate Overture *vdm2c* generated c-code into TASTE workflow (production)

Model-level integration of Overture into TASTE



General idea: execute a VDM operation as an external call in a SDL model

1. Asn1scc: all TASTE ASN.1 datatypes are converted into VDM data types
2. OpenGEODE simulator connects to Overture "remote call" API over a socket
3. OpenGEODE converts TASTE data values to and from VDM data values

From ASN.1 to VDM data types (1)

ASN.1 data type in TASTE

```
TypeC ::= INTEGER (0..255)
```

VDM data type in Overture

```
types
public TypeC = int
  inv x >= 0 and x <= 255
```

From ASN.1 to VDM data types (2)

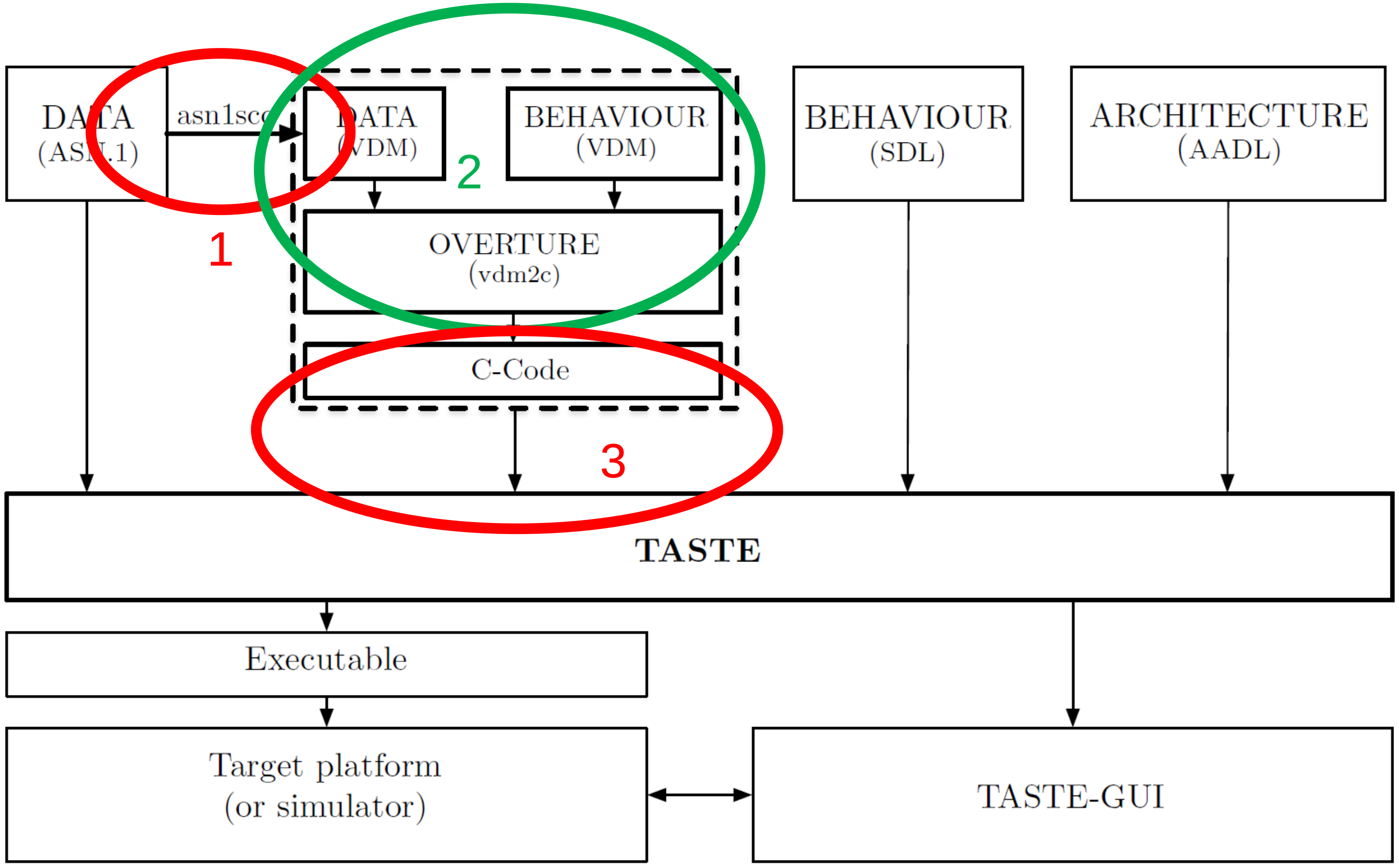
ASN.1 data type in TASTE

```
TASTE-Dataview DEFINITIONS ::=
BEGIN
IMPORTS T-Boolean FROM TASTE-BasicTypes;
Rover-State ::= SEQUENCE (SIZE(4)) OF REAL (0.0 .. 1000.0)
END
```

VDM data type in Overture

```
class TASTE_Dataview
types
    public Rover_State = seq of real
    inv x == len x = 4
end TASTE_Dataview

class TASTE_BasicTypes
types
    public T_Boolean = bool
end TASTE_BasicTypes
```

Code-level integration of Overture into TASTE



1. asn1scc : all TASTE ASN.1 datatypes are converted into VDM data types
2. vdm2c generates c-code from VDM++ models in a proprietary native format
3. integrate generated c-code into TASTE (automatic mapping to and from ASN.1)

vdm2c: representing VDM datatypes in C

```
typedef enum {
  VDM_INT, VDM_NAT, VDM_NAT1, VDM_BOOL, VDM_REAL,
  VDM_RAT, VDM_CHAR, VDM_SET, VDM_SEQ, VDM_MAP,
  VDM_PRODUCT, VDM_QUOTE, VDM_RECORD, VDM_CLASS
} vdmtype;

typedef union TypedValueType {
  void* ptr;              // VDM_SET, VDM_SEQ, VDM_CLASS,
                          // VDM_MAP, VDM_PRODUCT
  int intVal;             // VDM_INT and INT1
  bool boolVal;           // VDM_BOOL
  double doubleVal;       // VDM_REAL
  char charVal;           // VDM_CHAR
  unsigned int uintVal;   // VDM_QUOTE
} TypedValueType;

struct TypedValue {
  vdmtype type;
  TypedValueType value;
};

struct Collection {
  struct TypedValue** value;
  int size;
};
```

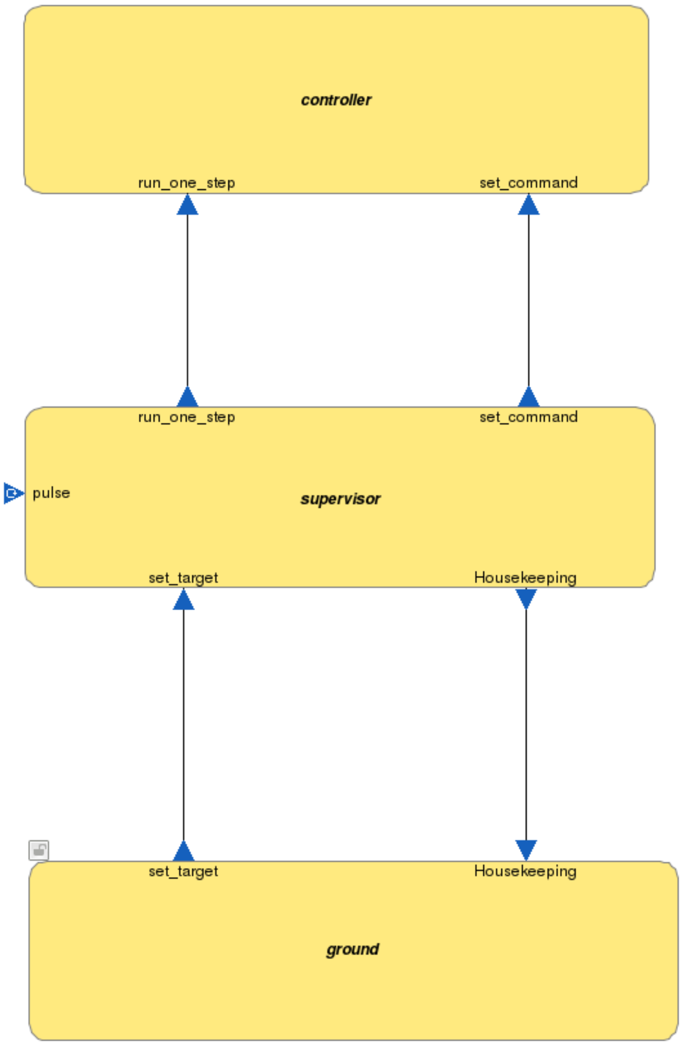
Marshalling vdm2c datatypes to and from ASN.1

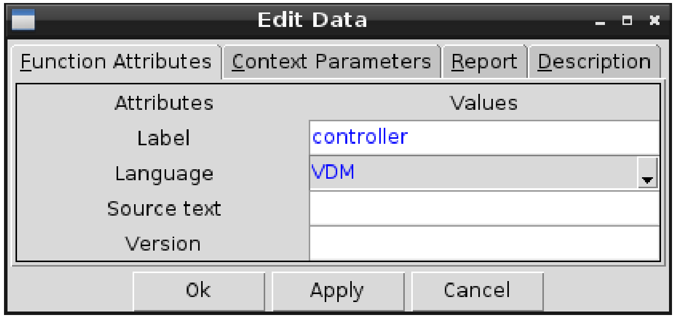
```
typedef asn1Sccint TypeC;
```

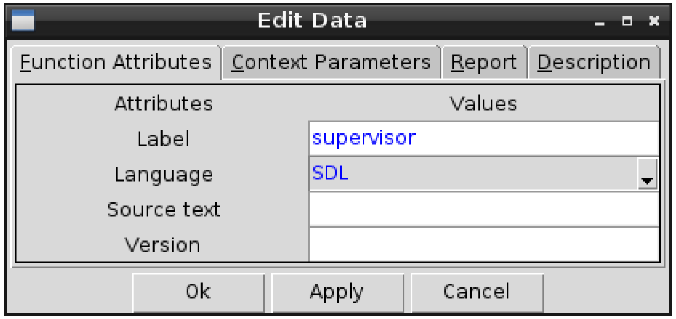
```
void Convert_TypeC_from_VDM_to_ASN1SCC
  (asn1SccMInt *ptrASN1SCC, TVP VDM)
{
  (*ptrASN1SCC) = (asn1SccSint)((VDM))->value.intVal;
}

void Convert_TypeC_from_ASN1SCC_to_VDM
  (TVP *ptrVDM, const asn1SccMInt *ptrASN1SCC)
{
  (*ptrVDM) = newInt((*ptrASN1SCC));
}
```

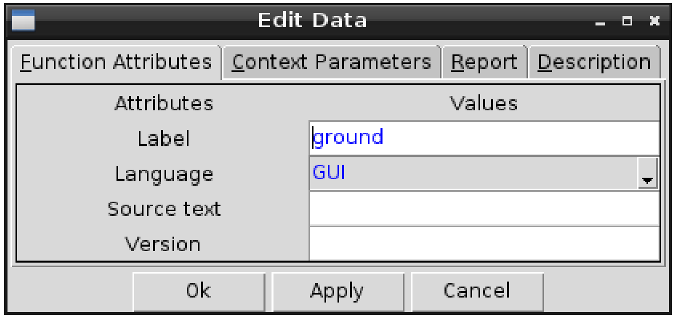
Tool support is available to automatically generate these marshalling functions

An example

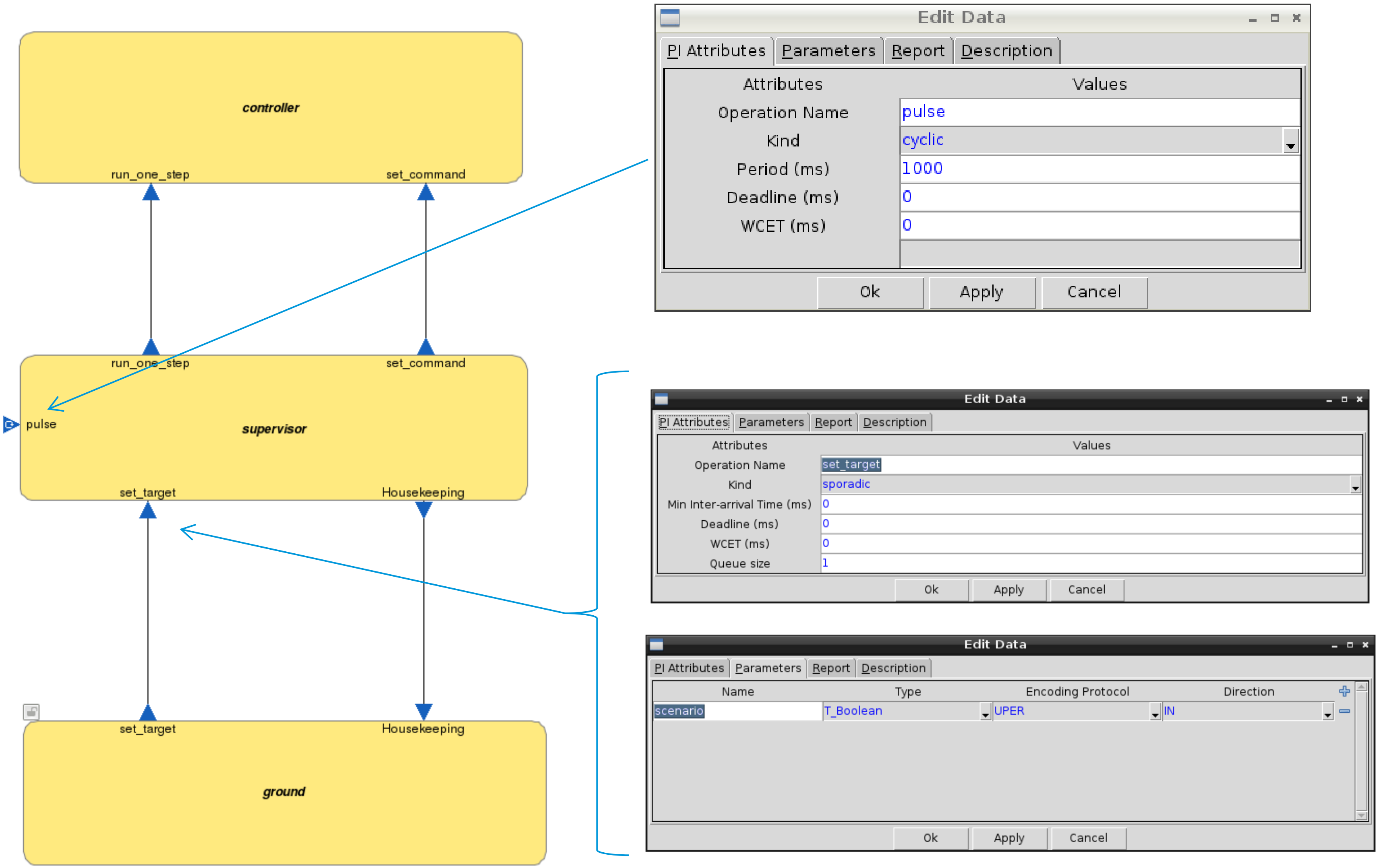

controller
run_one_step
set_command
run_one_step
set_command
pulse
supervisor
set_target
Housekeeping
set_target
Housekeeping
ground


Edit Data
Function Attributes
Context Parameters
Report
Description
Attributes
Values
Label
controller
Language
VDM
Source text
Version
Ok
Apply
Cancel

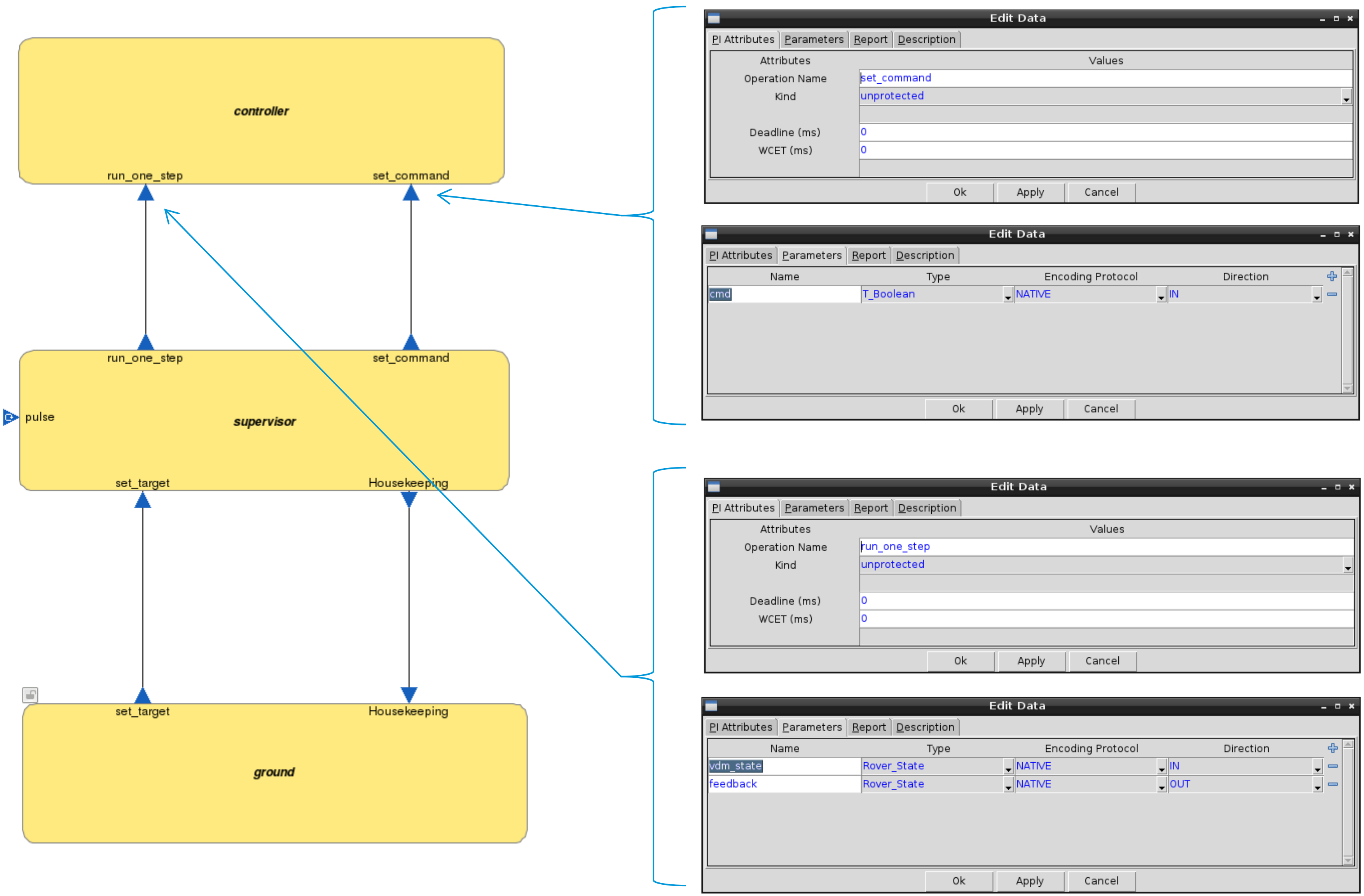

Edit Data
Function Attributes
Context Parameters
Report
Description
Attributes
Values
Label
supervisor
Language
SDL
Source text
Version
Ok
Apply
Cancel


Edit Data
Function Attributes
Context Parameters
Report
Description
Attributes
Values
Label
ground
Language
GUI
Source text
Version
Ok
Apply
Cancel

An example

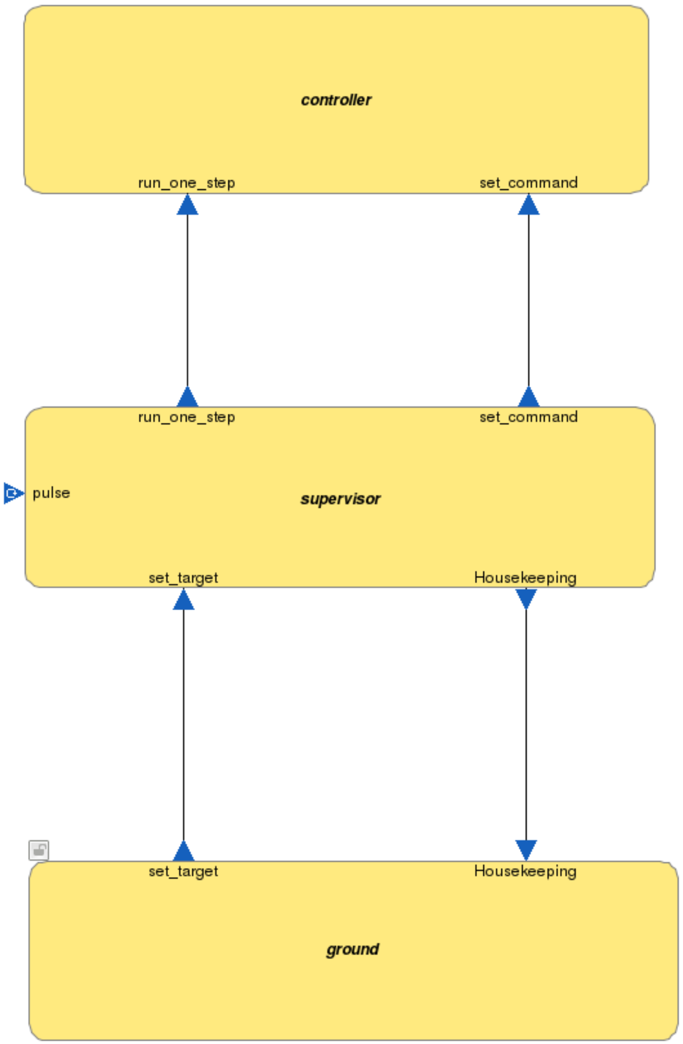

controller
run_one_step
set_command
run_one_step
set_command
pulse
supervisor
set_target
Housekeeping
set_target
Housekeeping
ground
Edit Data
PI Attributes
Parameters
Report
Description
Attributes
Values
Operation Name
pulse
Kind
cyclic
Period (ms)
1000
Deadline (ms)
0
WCET (ms)
0
Ok
Apply
Cancel
Edit Data
PI Attributes
Parameters
Report
Description
Attributes
Values
Operation Name
set_target
Kind
sporadic
Min Inter-arrival Time (ms)
0
Deadline (ms)
0
WCET (ms)
0
Queue size
1
Ok
Apply
Cancel
Edit Data
PI Attributes
Parameters
Report
Description
Name
Type
Encoding Protocol
Direction
scenario
T_Boolean
UPER
IN
Ok
Apply
Cancel

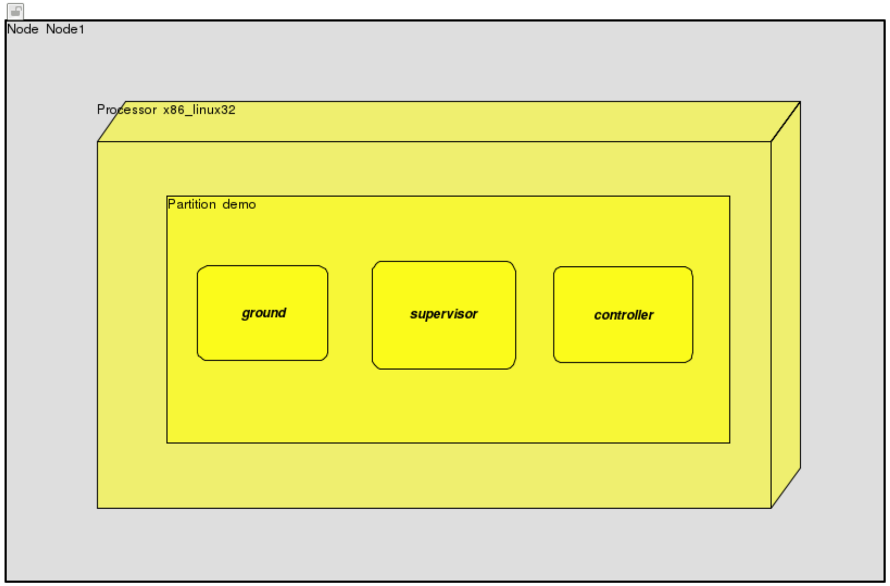
An example



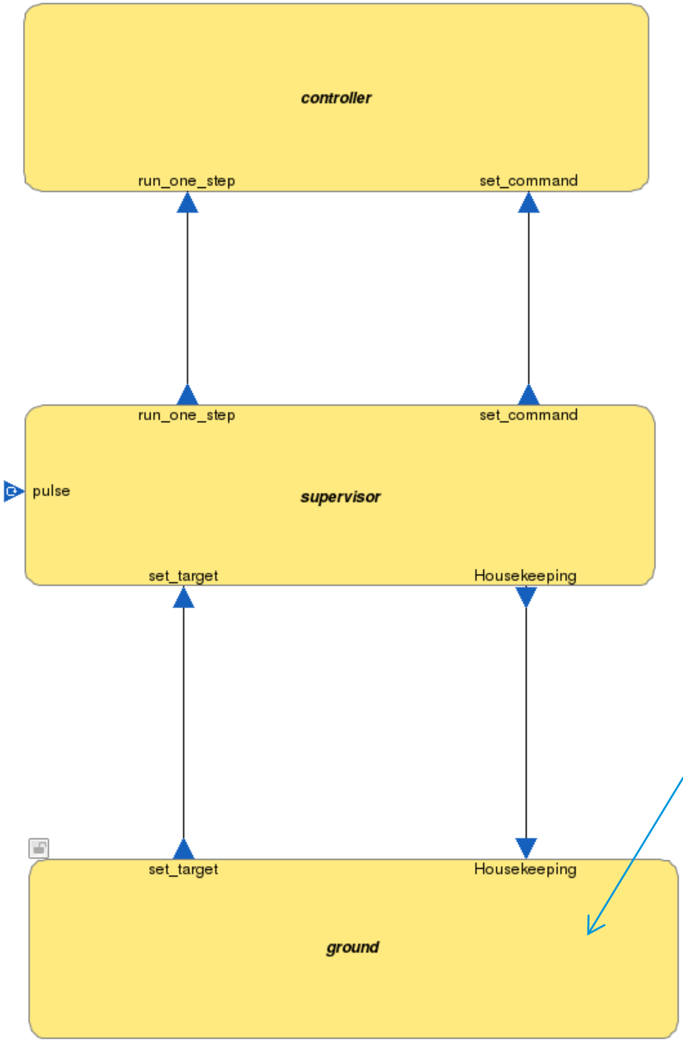
controller
run_one_step
set_command
run_one_step
set_command
pulse
supervisor
set_target
Housekeeping
set_target
Housekeeping
ground
Edit Data
PI Attributes
Parameters
Report
Description
Attributes
Values
Operation Name
set_command
Kind
unprotected
Deadline (ms)
0
WCET (ms)
0
Ok
Apply
Cancel
Edit Data
PI Attributes
Parameters
Report
Description
Name
Type
Encoding Protocol
Direction
cmd
T_Boolean
NATIVE
IN
Ok
Apply
Cancel
Edit Data
PI Attributes
Parameters
Report
Description
Attributes
Values
Operation Name
run_one_step
Kind
unprotected
Deadline (ms)
0
WCET (ms)
0
Ok
Apply
Cancel
Edit Data
PI Attributes
Parameters
Report
Description
Name
Type
Encoding Protocol
Direction
vdm_state
Rover_State
NATIVE
IN
feedback
Rover_State
NATIVE
OUT
Ok
Apply
Cancel

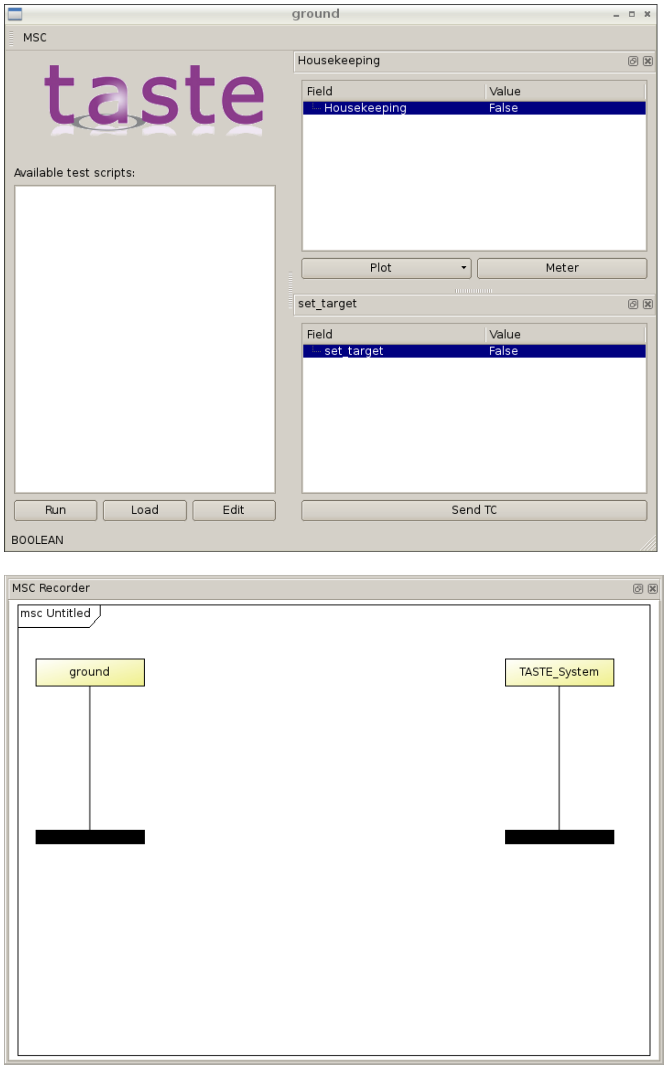
An example


controller
run_one_step
set_command
run_one_step
set_command
pulse
supervisor
set_target
Housekeeping
set_target
Housekeeping
ground

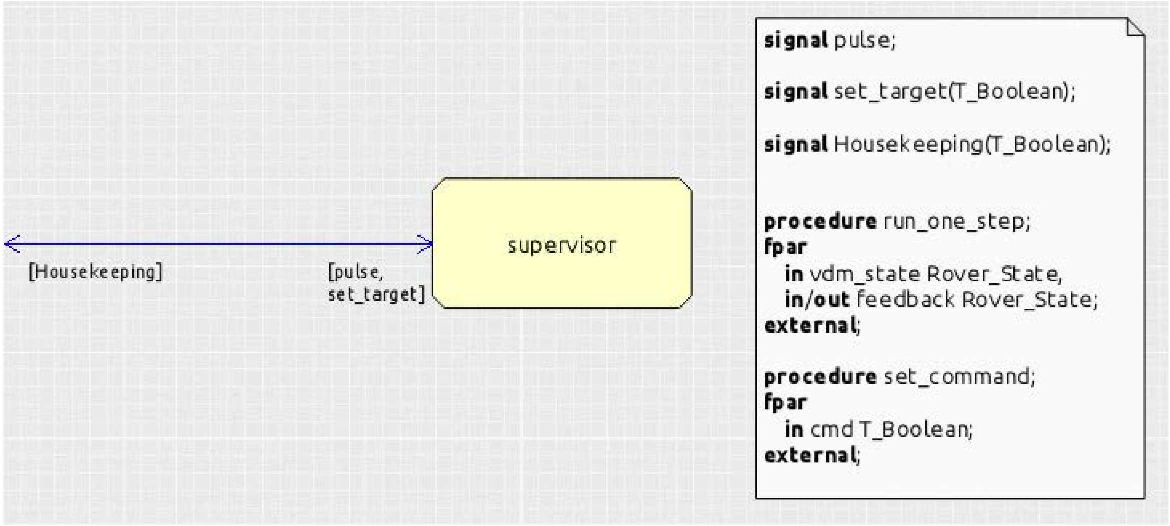

Node Node1
Processor x86_linux32
Partition demo
ground
supervisor
controller

An example

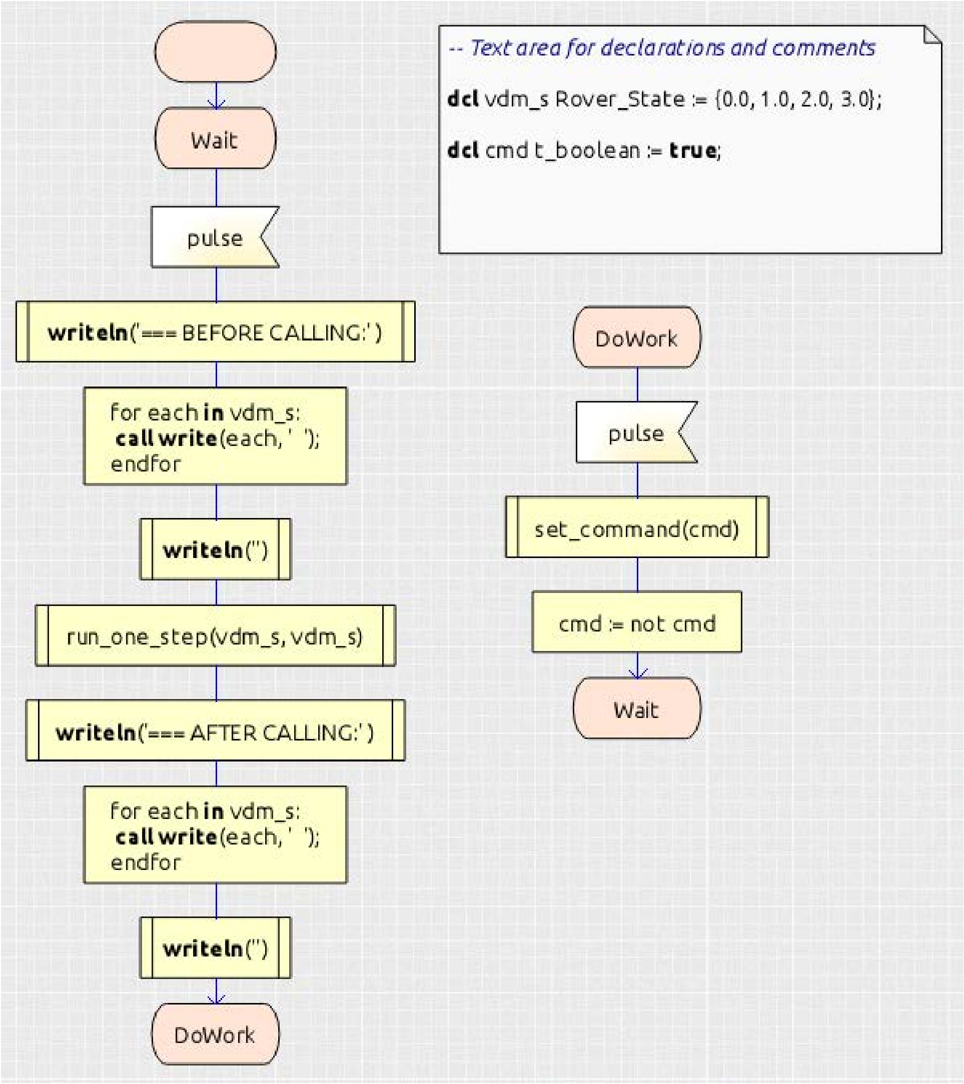

controller
run_one_step
set_command
run_one_step
set_command
pulse
supervisor
set_target
Housekeeping
set_target
Housekeeping
ground


ground
MSC
taste
Available test scripts:
Housekeeping
Field
Value
Housekeeping
False
Plot
Meter
set_target
Field
Value
set_target
False
Run
Load
Edit
Send TC
BOOLEAN
MSC Recorder
msc Untitled
ground
TASTE_System

SDL model of the supervisor



SDL model of the supervisor



VDM model of the controller (1)

```
class controller_Interface
operations
  public Startup: () ==> ()
  Startup () is subclass responsibility;

  public PI_run_one_step: TASTE_Dataview`Rover_State ==>
    TASTE_Dataview`Rover_State
  run_one_step (-) == is subclass responsibility;

  public PI_set_command: TASTE_BasicTypes`T_Boolean ==> ()
  set_command (-) == is subclass responsibility;

end controller_Interface
```

Tool support is available to automatically generate these interface definitions

VDM model of the controller (2)

```
class controller
  is subclass of controller_Interface

instance variables
  updateState : bool := false
operations
  public Startup: () ==> ()
  Startup () == updateState := true;

  public PI_run_one_step: TASTE_Dataview`Rover_State ==>
    TASTE_Dataview`Rover_State
  PI_run_one_step (vdm_state) ==
    if updateState then
      ( dcl newState : TASTE_Dataview`Rover_State :=
          [ vdm_state(1) + 1, vdm_state(2) + 2,
            vdm_state(3) + 3, vdm_state(4) + 4];
        return newState )
    else return vdm_state;

    public PI_set_command: TASTE_BasicTypes`T_Boolean ==> ()
    PI_set_command (cmd) == updateState := cmd;

end controller
```

User specifies the required behavior in VDM then uses *vdm2c* to generate c-code

Bringing it all together (1) – TASTE, ASN.1, VDM

```
#include "Vdm_ASN1_Types.h"
#include "controller.h"

static TVP controller;

void controller_startup()
{
  controller = _Z10controllerEV(NULL);
  CALL_FUNC(controller, controller, controller,
    CLASS_controller__Z7StartupEV);
}

void controller_PI_run_one_step (
  const asn1SccRover_State *IN_vdm_state,
  asn1SccRover_State *OUT_feedback )
{
  TVP ptr_vdm_state = NULL;
  Convert_Rover_State_from_ASN1SCC_to_VDM
    (&ptr_vdm_state, IN_vdm_state);
  TVP vdm_OUT_feedback;
  vdm_OUT_feedback = CALL_FUNC
    (controller, controller, controller, 1, ptr_vdm_state);
  Convert_Rover_State_from_VDM_to_ASN1SCC
    (OUT_feedback, &vdm_OUT_feedback);
}

void controller_PI_set_command(const asn1SccT_Boolean *IN_cmd)
{
    TVP ptr_cmd = NULL;
    Convert_T_Boolean_from_ASN1SCC_to_VDM(&ptr_cmd, IN_cmd);
    CALL_FUNC(controller, controller, controller, 2, ptr_cmd);
}
```

Tool support is available to automatically generate this glue code

Bringing it all together (1) – build and execute

```
=== BEFORE CALLING: 0   1   2   3
=== AFTER CALLING:  1   3   5   7
=== BEFORE CALLING: 1   3   5   7
=== AFTER CALLING:  2   5   8   11
=== BEFORE CALLING: 2   5   8   11
=== AFTER CALLING:  2   5   8   11
=== BEFORE CALLING: 2   5   8   11
=== AFTER CALLING:  3   7   11   15
=== BEFORE CALLING: 3   7   11   15
=== AFTER CALLING:  3   7   11   15
=== BEFORE CALLING: 3   7   11   15
=== AFTER CALLING:  4   9   14   19
```

In summary, we have fully automated:

- TASTE ASN.1 datatypes are converted into their VDM counterparts
- VDM interface skeletons are generated from AADL models
- ASN.1 marshalling functions are generated compatible with vdm2c target code
- Glue code is generated to integrate vdm2c target code easily into TASTE
- TASTE builds the (heterogeneous) binary application

The only manual steps the user has to perform are
(1) write VDM spec, (2) execute vdm2c and (3) run build-script.sh

Conclusions and future work

Main findings

- We have shown that the integration of Overture into TASTE is feasible
- All TASTE ASN.1 datatypes can be translated into their VDM counterparts
- Glue code and ASN.1 marshalling functions can be generated for a subset of VDM data types (integer, real, bool, seq of) → *vdm2c v0.0.2*
- Complexity of the integration can be hidden entirely by automation

Next steps (short term)

- Extend the glue code and marshalling generator (follow vdm2c evolution)
- Allow headless build (vdm2c executed as part of TASTE build process)

Next steps (long term)

- Make vdm2c aware of ASN.1 / static memory allocation (qualified code)
- Embed ASN.1 capability directly into Overture interpreter (remote api)