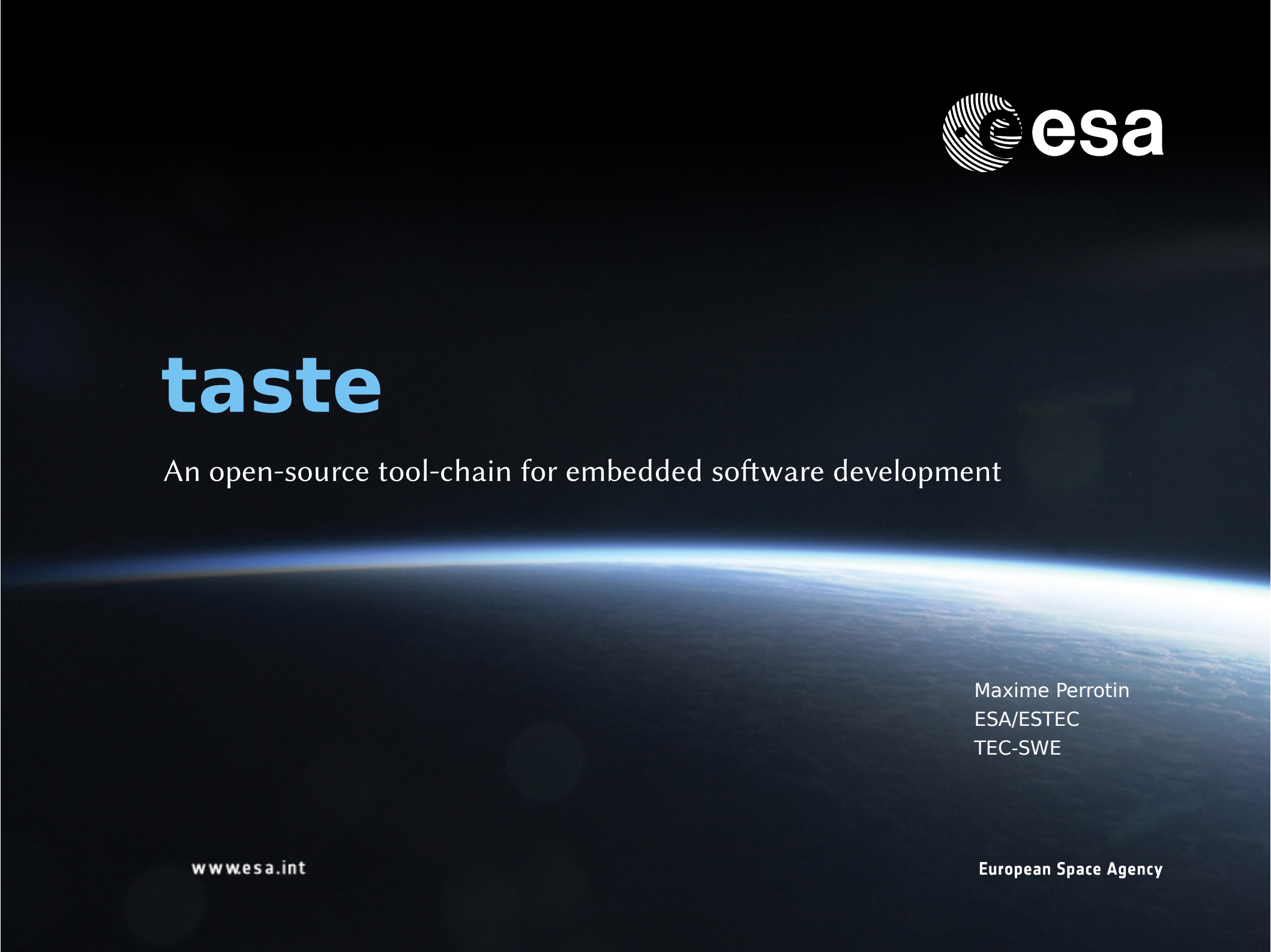


taste

An open-source tool-chain for embedded software development

Maxime Perrotin
ESA/ESTEC
TEC-SWE

www.esa.int

European Space Agency

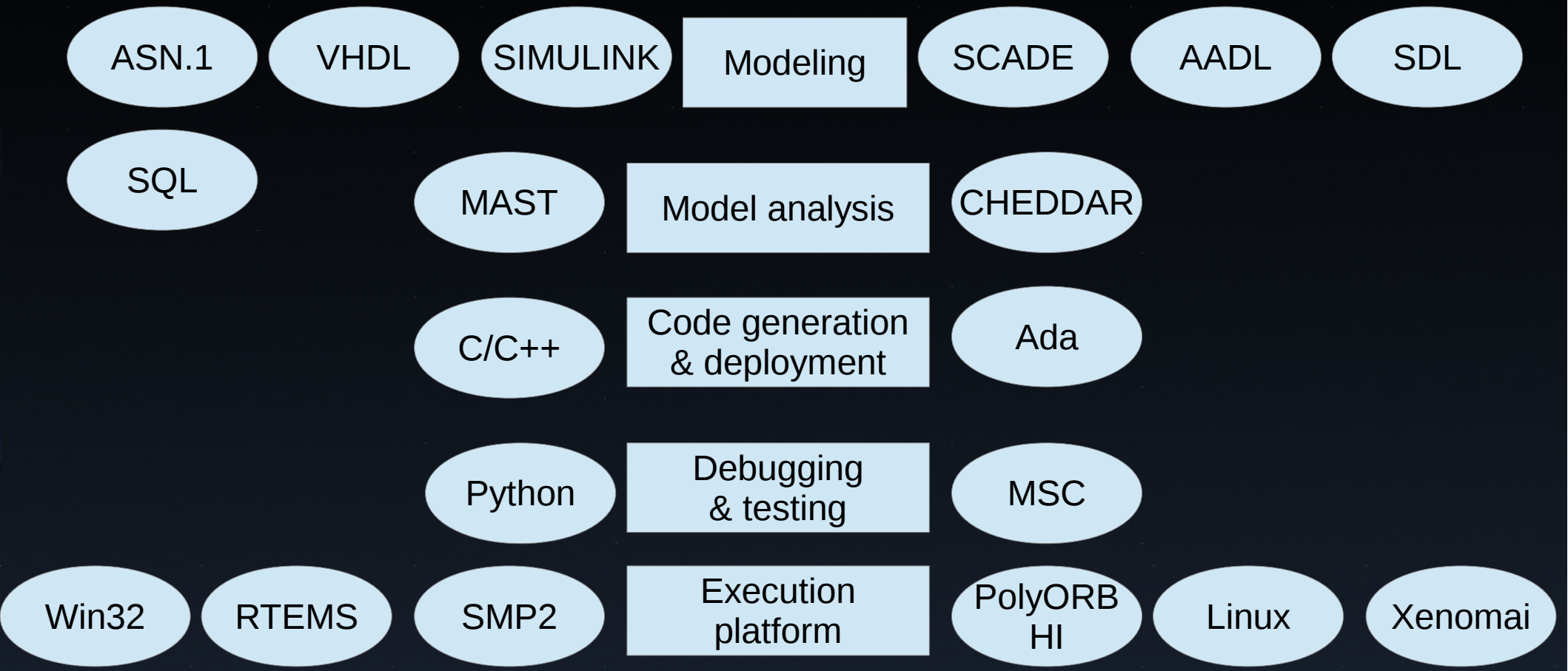
Introduction - what is TASTE ?

- A tool-chain targeting heterogeneous, embedded systems, using a model-centric development approach
- A laboratory platform for experimenting new software-related technologies, based on free, open-source solutions
- A process supporting the creation of systems using formal models and automatic code generation

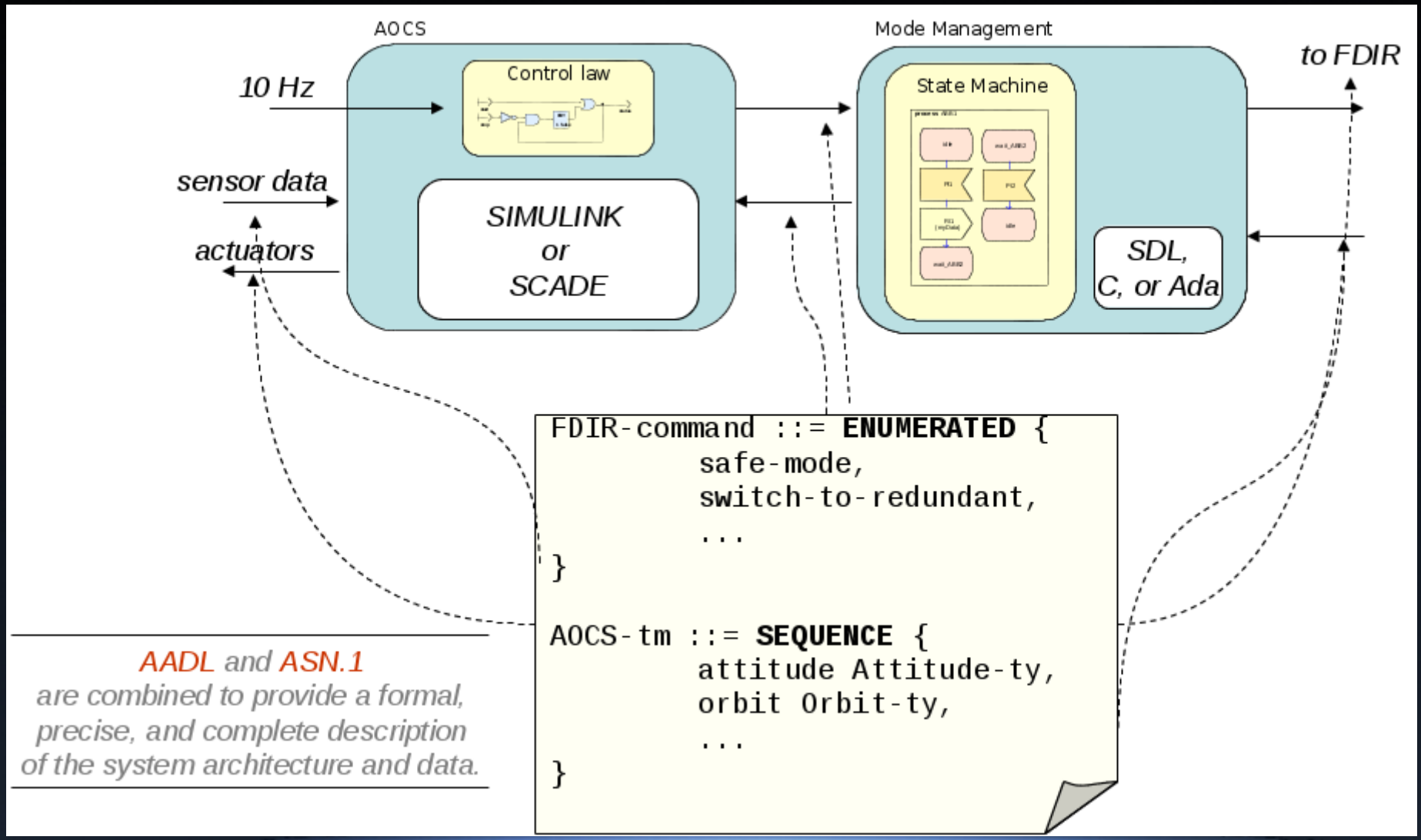
Main targets and philosophy

- Use DSL – select the most appropriate language to solve a problem
- But do not reinvent the wheel : use existing languages and tools
  - Mature languages with long-term support
  - Interoperable tools, limiting vendor lock-in
- Not all languages however
  - We target safe systems

Supported technologies



Quick overview



Status at a glance

- **Mature tools with commercial support**
  - AADL to Ada and C compilers
  - ASN.1 compiler (Ada and C) and tools
  - SDL editor with Ada code generator
- **Maturing tools (prototype-level)**
  - GUI & Testing Framework
  - Model verification tools (Cheddar, Mast, Model Checker)
  - Hardware-Software co-engineering package
  - Support for SMP2

Distribution

- Bundled in a virtual machine

  → full Debian Linux installation, with all dependencies installed

  → Update mechanism triggered from the desktop
- Manual installation is possible (instructions on the Wiki) – Ask for help.
- Documentation and reference card on the desktop

  → Print the reference card !
- Linux-based, but a Windows GUI prototype exists

- The following slides show how TASTE can be used for
  - Software quick prototyping
  - Helping review process and improve software quality

- In addition most of TASTE features are presented

Target : reactive and discrete systems

- Communication :
  - Message exchanges between the system and its environment
  - Asynchronous and synchronous interactions
- Algorithms, GUIs
- Databases
- Wide range of applications
  - Safety and mission-critical communicating systems
  - Real-time applications (embedded systems)
- Wide range of architectures
  - X86 (Linux, Win32, Xenomai, FreeBSD), SPARC (Leon), ARM
  - 32 and 64 bits

TASTE as a support for reviews

- Model the requirements and/or the design
  - **To help understanding a specification or a design**
    - What is the system intended to do?
    - Who are the actors (scope)?
    - What is the expected behaviour?
    - What is the data ?
  - **To get an executable representation of the system**
    - Early and independent testing ⟶ valuable inputs for a review or project support
  - **To improve the quality of the specification**
    - Formal models enable various verifications,
    - Detect ambiguities ⟶ less risk of having incomplete requirements

Tools help finding many classes of errors in specifications

- Ambiguities with **data** – often no shared data dictionnary. Inconsistencies with namings, semantics, scope...
- Missing **interface** information (behaviour, off-nominal handling, parameter constraints...)
- **Sequencing** (dynamic) issues – what is done in what order, etc.
- **Completeness** of paths

TASTE process

1) Describe the system logical architecture and interfaces with ASN.1 and AADL

2) Generate code skeletons and write the applicative code or models

3) Capture the system hardware and deployment

4) Verify models

5) Build the system and download it on target

6) Monitor and interact with the system at run-time

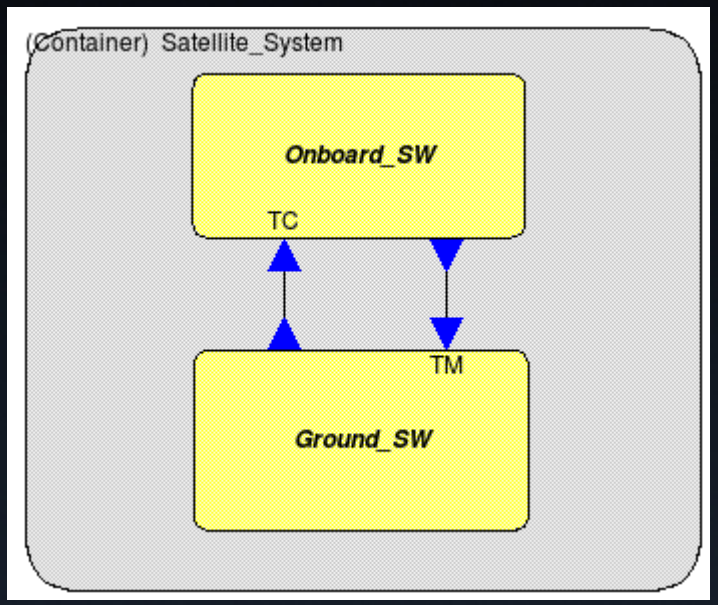
Formal languages

- **TASTE relies on formal languages** :
  - ASN.1 and AADL to capture the software architecture and data
  - SDL, Simulink, SCADE, C, Ada, VHDL, ... to capture the software behaviour
  - MSC and Python to test
- **Combine graphical AND textual notations**
  - If anything goes wrong, human can fix textual syntax
  - Diagrams for easier understanding
  - But some information is textual by nature
- **Avoid languages with weak semantics or syntax**

- Architecture Analysis and Design Language, a standard from SAE
- Used to model the **logical and the physical architecture** of the system
- Textual and graphical representations
- Used in TASTE to capture the system structure, interfaces, hardware and deployment.

TASTE interface view

- Two entities : **containers** and **functions**, to capture the system logical architecture



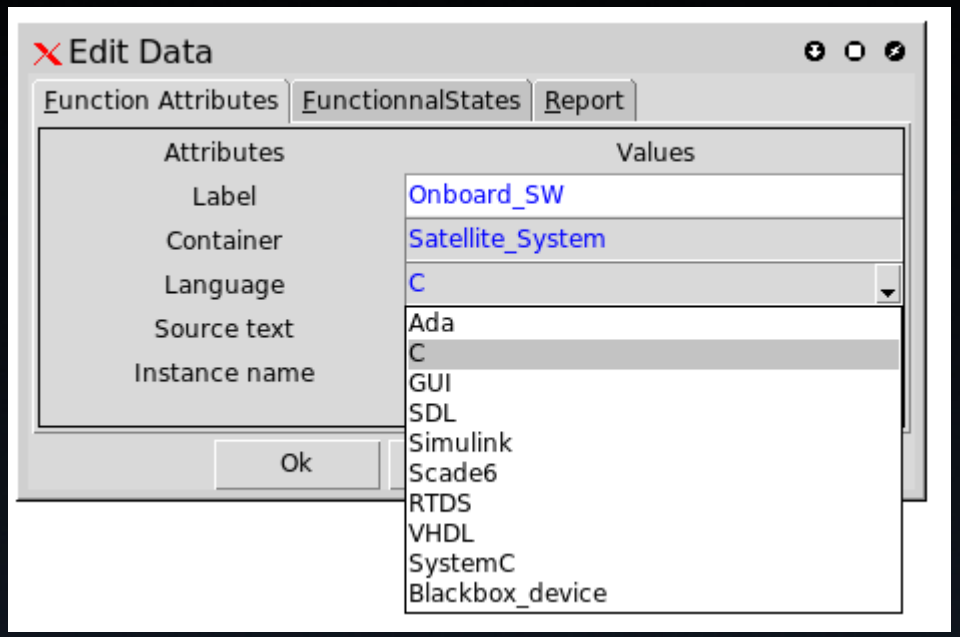
Function

- A function is a terminal level entity. It has a behaviour that can be triggered through a set of **provided interfaces**.

- All interfaces of a function have visibility and control access on the function's internal data (static data).

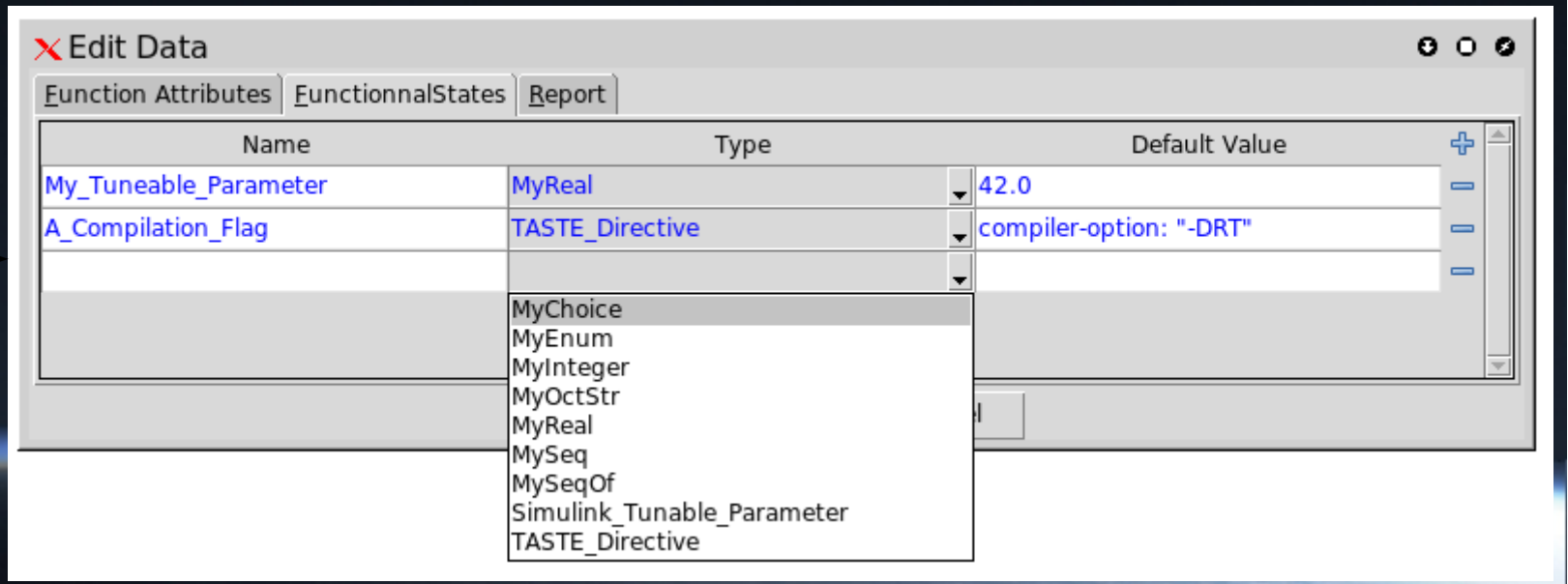
- With one exception, the interfaces of a function are mutually exclusive, and run to completion (it is not possible to execute concurrently two interfaces of a function, as they share state data).

Properties of a function



The implementation language

Optional « context parameters »



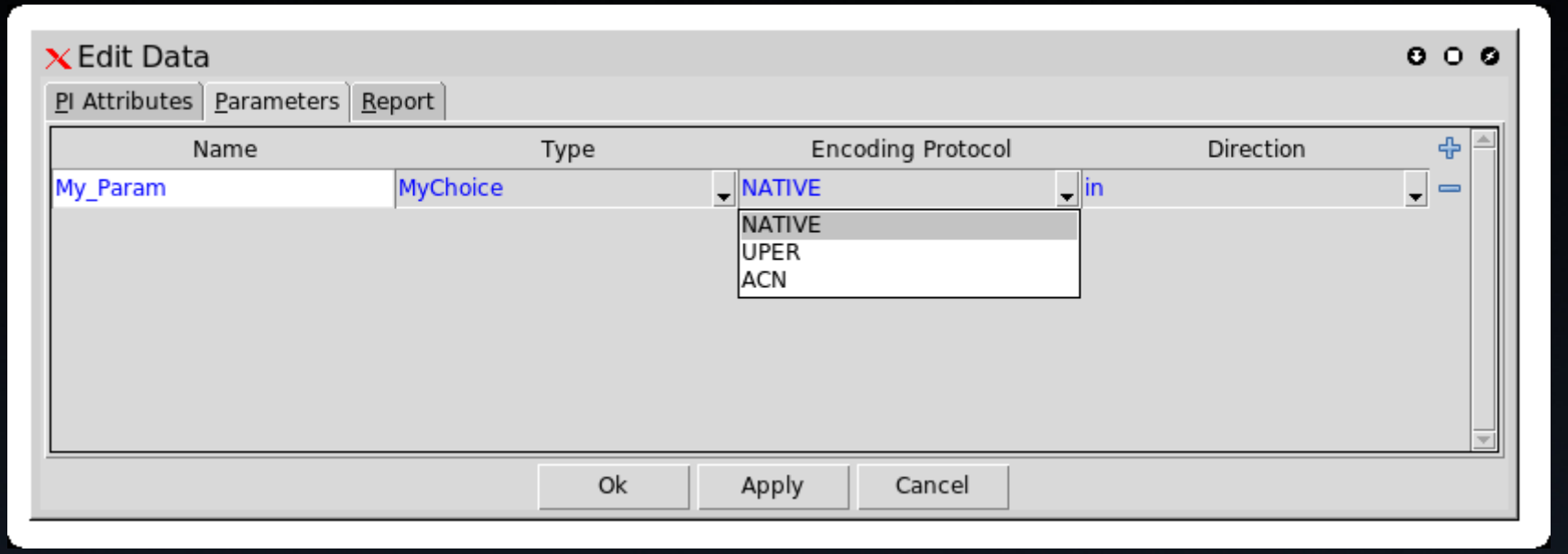
Context Parameters

- The « Functional State » tab offers a space for flexibility :
  - **Context parameters** allow defining constants at model level and make them accessible from user code
    - Support for C, Ada and Simulink (instructs code generator to generate « tuneable parameters », which are global variables)
    - Value can be generated from an external source
  - **TASTE directives** are used to fine-tune the build process with additional properties (e.g. compilation or link flags that are specific to a piece of code)
    - Used to integrate Simulink code when it requires special defines (-DRT, -DUSE_RTMODEL)
    - When a property proves usefulness, it gains a dedicated entry in the GUI

Provided and required interfaces

- A provided interface (PI) is a service offered by a function. It can be
  - **Periodic**, in which case it does not take any parameter, and is used to handle cyclic tasks
  - **Sporadic (**or **asynchronous**) and optionally carry a parameter. The actual execution time is decided by the real-time scheduler (call is *deffered*)
  - **Synchronous**, with or without **protection** and optionally carry parameters (in and out)
    - The protection is a semaphore (in C) or a protected object (in Ada) preventing concurrent execution of several interfaces of the same function.
    - Use unprotected interface to implement e.g. « getter » functions
    - Caller blocks on execution (call is *immediate*) – Just like a direct function call.
    - At runtime, synchronous functions execute in the caller's thread space.

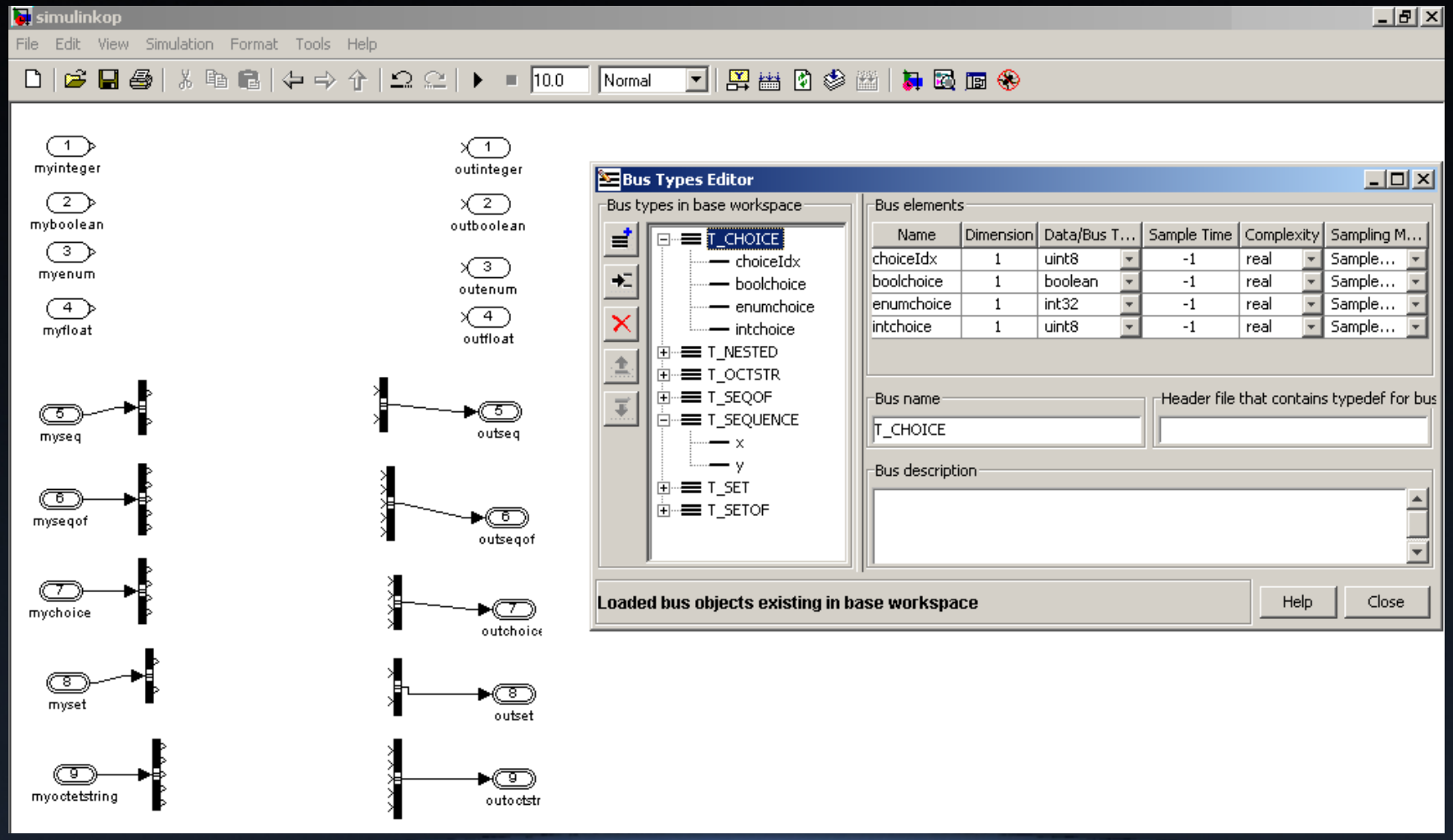
Function parameters



Each parameter has a type (from the ASN.1 model),
a **direction** (in or out), and an **encoding protocol :**

Native : means memory dump – no special treatment
UPER : compact binary encoding
ACN : user-defined encoding

Skeleton example : Simulink



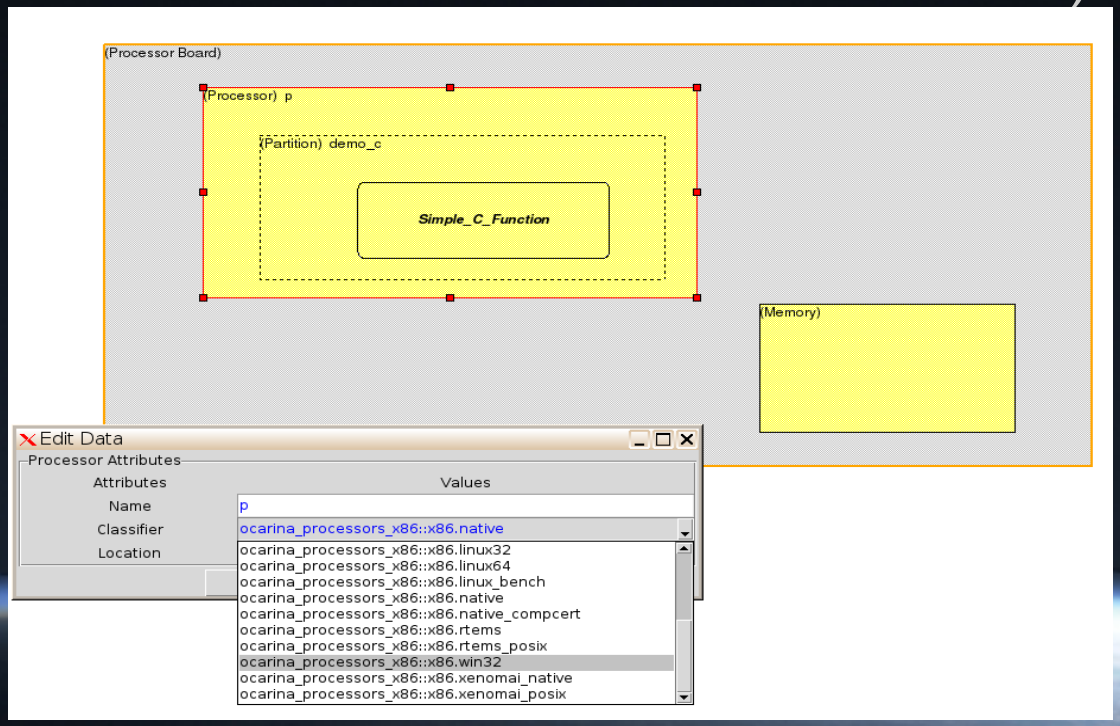
Build script

- A build script for the system is generated automatically : *build-script.sh*
- It may need to be tuned to select the runtime (C or Ada) or for advanced options

- Before calling the script, a deployment diagram has to be filled

Deployment view

- Map functions on hardware
- Centralized and distributed systems
- Can add buses, drivers.. Extensible (every component is described in an AADL file)



SDL, MSC and ASN.1

- **SDL : Specification and Description Language (ITU-T standard)**
  - SDL is a state machine modelling language
- **ASN.1 : Abstract Syntax Notation One (ISO, ITU-T)**
  - Describe data types and constraints
  - and data physical representation
- **MSC : Message Sequence Charts (ITU-T) a.k.a. Sequence diagrams**
  - A behavioural specification language and tracing tool

ASN.1

- International, widely used standard (ISO and ITU-T)
- Simple text notation for precise and complete **data type description**
- Real added value : the physical encoding rules (compact binary encoding, endianness-neutral, but also XML encoding, legacy encoding specifications).
- Separate the encoding rules from the types specification

ASN.1 – basic types

INTEGER

→ My-int ::= INTEGER (0..7)

value My-int ::= 5

REAL

→ My-real ::= REAL (10.0 .. 42.0)

BOOLEAN

ENUMERATED

→ My-enum ::= ENUMERATED { hello, world }

OCTET STRING

→ My-string ::= OCTET STRING (SIZE (0..255))

value My-string::= 'DEADBEEF'H

BIT STRING

→ My-bitstring ::= BIT STRING (SIZE (10..12))

value My-bitstring ::= '00111000110'B

ASN.1 – complex types

- SEQUENCE
  - → My-seq ::= **SEQUENCE** {
    x My-int,
    y My-enum OPTIONAL
    }
    value My-seq::= { x 5 }
- CHOICE
  - → My-choice ::= CHOICE {
    choiceA My-real,
    choiceB My-bitstring
    }
    value My-choice ::= choiceA : 42.0
- SEQUENCE OF
  - → My-seq ::= SEQUENCE (SIZE (0..5)) OF BOOLEAN
    value My-seq:= { 1, 2 ,3 }
- SET / SET OF

ASN.1 benefits – CFDP example

Field	Length (bits)	Values	Comment
Version	3	'000'	For the first version.
PDU type	1	'0' — File Directive '1' — File Data	
Direction	1	'0' — toward file receiver '1' — toward file sender	Used to perform PDU forwarding.
Transmission Mode	1	'0' — acknowledged '1' — unacknowledged	
CRC Flag	1	'0' — CRC not present '1' — CRC present	
Reserved for future use	1	set to '0'	
PDU Data field length	16		In octets.
Reserved for future use	1	set to '0'	
Length of entity IDs	3		Number of octets in entity ID less one; i.e., '0' means that entity ID is one octet. Applies to all entity IDs in the PDU header.
Reserved for future use	1	set to '0'	
Length of Transaction sequence number	3		Number of octets in sequence number less one; i.e., '0' means that sequence number is one octet.
Source entity ID	variable		Uniquely identifies the entity that originated the transaction.
Transaction sequence number	variable		Uniquely identifies the transaction, among all transactions originated by this entity.
Destination entity ID	variable		Uniquely identifies the entity that is the final destination of the transaction's metadata and file data.

These fields are not application semantics! They concern the binary encoding rules of the PDUs and should not be mixed with the protocol useful information.

CFDP in ASN.1

- Keep only application-semantic data
- Tools will generate encoders and decoders to add the other fields

```
Packet-ty ::=  SEQUENCE {
        version                         Version-ty,
        direction                       Direction-ty,
        transmission-mode               Transmission-mode-ty,
        crc-flag                        CRC-flag-ty,
        source-entity-id                Entity-id-ty,
        transaction-sequence-number Transaction-sequence-number-ty,
        destination-entity-id           Entity-id-ty,
        data                            Datafield-ty
}

Version-ty ::= INTEGER (0..7)

Direction-ty ::= ENUMERATED { toward-file-receiver, toward-file-sender
}
```

ASN.1 : The exoskeleton case study

hardware interface (1)

Communication & State Management

(SDL/RTDS)

hardware interface (2)

Exoskeleton

(Sensors)

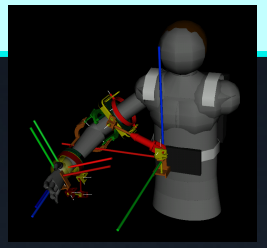
software interface

Control law

(Simulink)

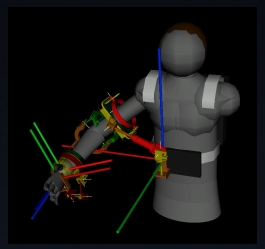
Robotic arm

(Actuators)



Solution : ASN.1 and ACN

One logical model for the end user (in ASN.1),
And one separate model describing the encoding
→ No need to worry about endianness, fields ordering, etc.



Edit and load Data View

dataview.asn

```
-- Output types

VR-Model-Output ::= SEQUENCE {
    x1 REAL (-1000 .. 1000),
    y1 REAL(-1000 .. 1000),
    z1 REAL(-1000 .. 1000),
    p1 REAL(-1000 .. 1000),
    x2 REAL(-1000 .. 1000),
    y2 REAL(-1000 .. 1000),
    z2 REAL(-1000 .. 1000),
    p2 REAL(-1000 .. 1000),
    x3 REAL(-1000 .. 1000),
    y3 REAL(-1000 .. 1000),
    z3 REAL(-1000 .. 1000),
    p3 REAL(-1000 .. 1000),
    j-rad SEQUENCE (SIZE(16)) OF REAL (-1000 .. 1000)
}
```

+

dataview-uniq.acn

```
/*Output types*/
VR-Model-Output []{
        j-rad [size 16] {
                dummy [encoding IEEE754-1985-64, endianness little]
        },
        p1 [encoding IEEE754-1985-64, endianness little] ,
        p2 [encoding IEEE754-1985-64, endianness little] ,
        p3 [encoding IEEE754-1985-64, endianness little] ,
        x1 [encoding IEEE754-1985-64, endianness little] ,
        x2 [encoding IEEE754-1985-64, endianness little] ,
        x3 [encoding IEEE754-1985-64, endianness little] ,
        y1 [encoding IEEE754-1985-64, endianness little] ,
        y2 [encoding IEEE754-1985-64, endianness little] ,
        y3 [encoding IEEE754-1985-64, endianness little] ,
        z1 [encoding IEEE754-1985-64, endianness little] ,
        z2 [encoding IEEE754-1985-64, endianness little] ,
        z3 [encoding IEEE754-1985-64, endianness little]
}
```

C Tab Width: 8 Ln 1, Col 1 INS

Our ASN.1 compiler

- Developped by Semantix (now Neuropublic) for ESA
- Free software (LGPL)
- Unique features – no competing tool :
  - Generates optimized C code (fast, low memory footprint)
  - Or SPARK/Ada code
  - No malloc, no system call
  - Automatically generates test cases for a given grammar
  - Generates ICDs documents in HTML format
  - Supports ACN for customized encodings (e.g. PUS format)
- Can be used independently from TASTE
- TASTE includes backends to give access to ASN.1 types to SDL, Simulink, SCADE, VHDL, SQL, Python

ACN – the basics

- ACN allows to specify legacy encodings
- It can be used to describe the binary format of PUS packets, leaving the interesting part only (payload data) in the ASN.1 side.

```
MySeq ::= SEQUENCE {
    alpha  INTEGER,
    gamma  REAL OPTIONAL
}
```

```
MySeq[] {
    alpha [],
    beta  BOOLEAN  [],
    gamma [present-when beta, encoding IEEE754-1985-64]
}
```

ACN – more examples

```
COLOR-DATA ::= CHOICE {
    green INTEGER (1..10),
    red    INTEGER (1..1000),
    blue   IA5String (SIZE(1..20))
}

MySeq ::= SEQUENCE {
    colorData COLOR-DATA
}
```

```
COLOR-TYPE [encoding pos-int, size 8]

MySeq [] {
    activeColor1 COLOR-TYPE [],
    activeColor2 COLOR-TYPE [],
    colorData    <activeColor1, activeColor2> []
    }

COLOR-DATA<COLOR-TYPE:type1, COLOR-TYPE:type2> [] {
        green [present-when type1==1  type2==10],
        red   [present-when type1==20 type2==20],
        blue  [present-when type1==50 type2==20]
    }
```

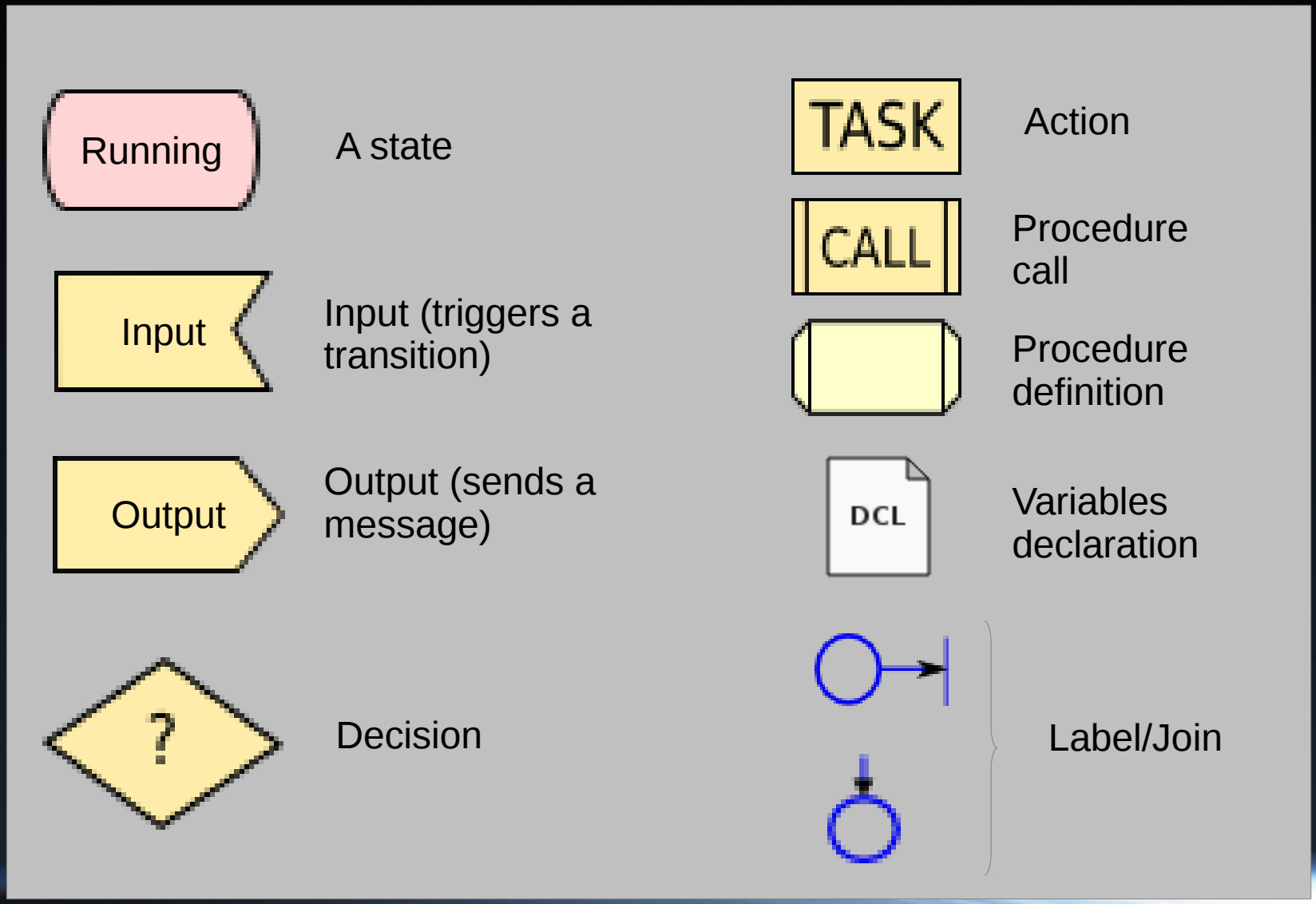
ACN - documentation

- User manual in the TASTE VM :

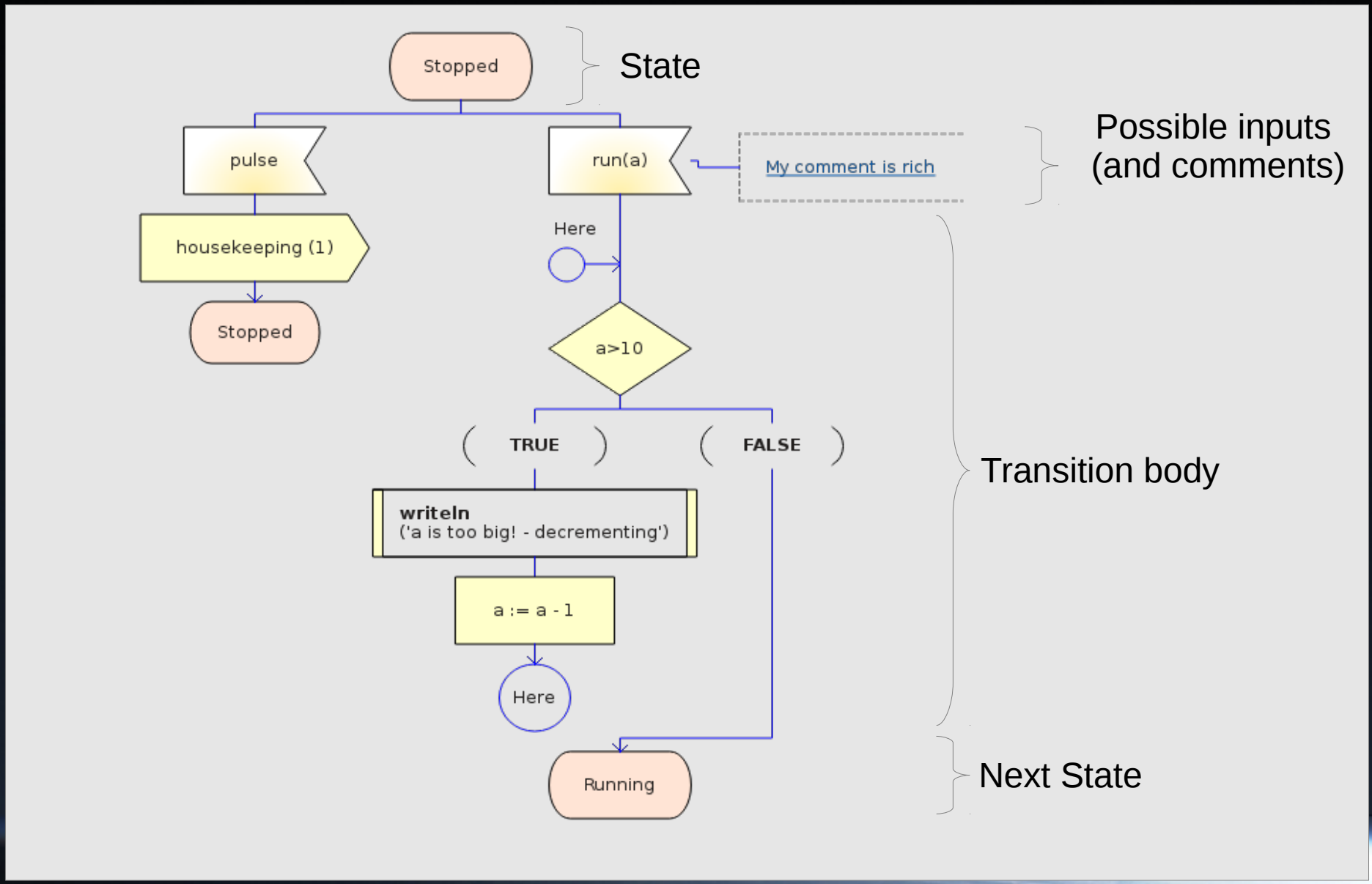
  `/home/assert/tool-src/doc/acn`

- Specification and Description Language (standard from ITU-T)
- A formal language for describing state-machines, graphically or textually.
- Easy to use, yet very powerful (manipulation of data, precise and complete semantics)
- Various mature commercial tools (e.g. RTDS)
- TASTE comes with an integrated SDL editor including an Ada code generator and natively supporting ASN.1 : **OpenGEODE**
  - Prototype level, under development
  - Free software, open source
  - Restricted to TASTE scope (embedded, real-time systems)
  - TASTE also supports commercial tools (ObjectGEODE, RTDS)

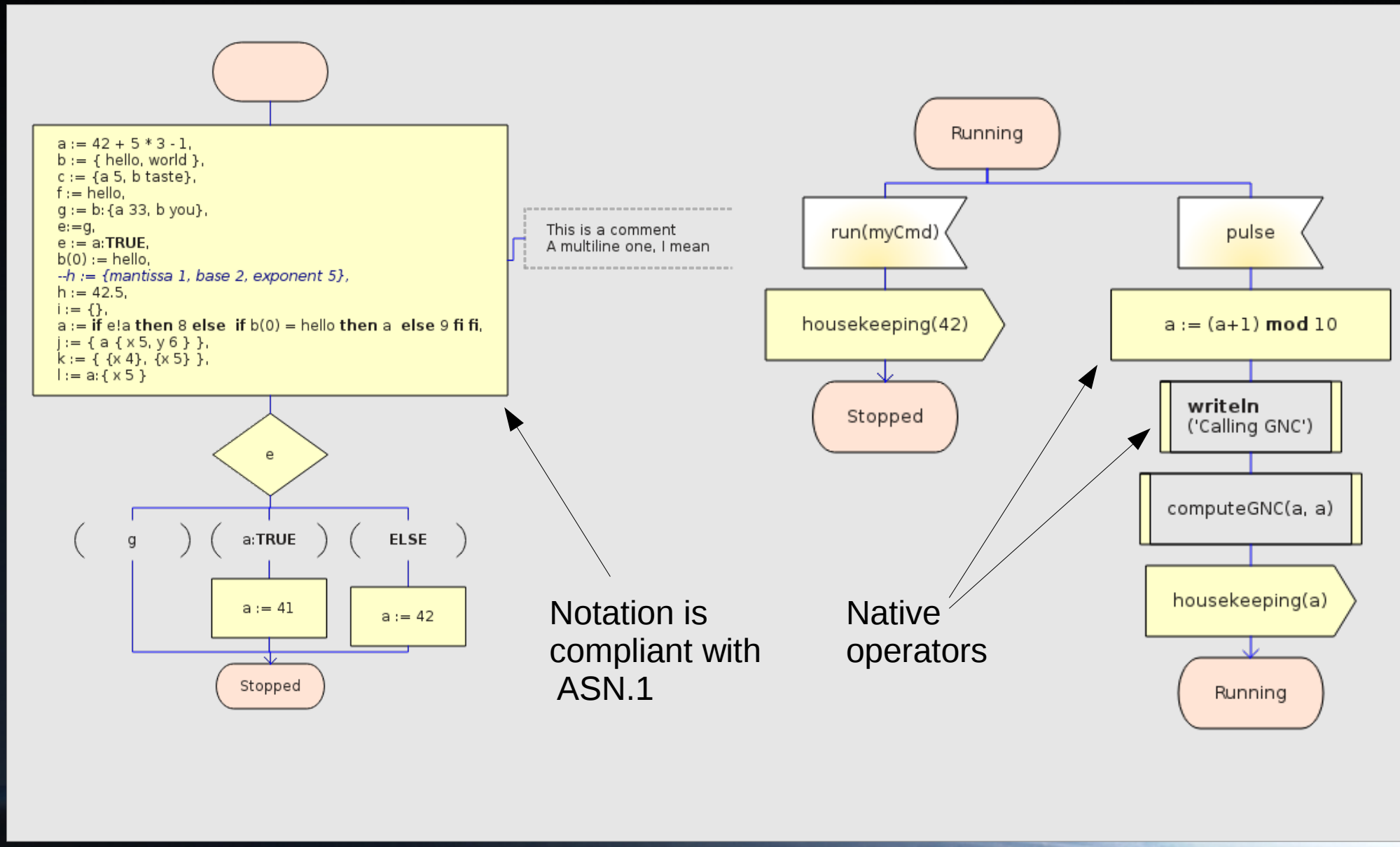
Major SDL elements for behavioural design



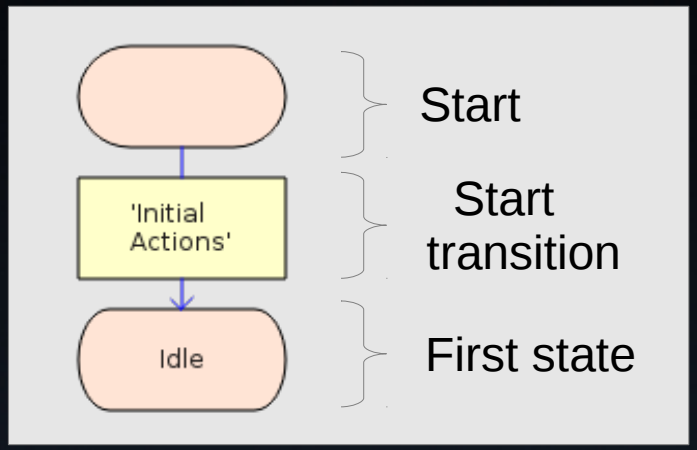
Typical transition diagram



Data manipulation (overview)

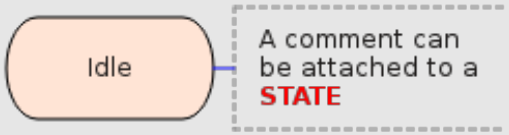


Start : initialization transition



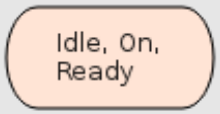
- A state machine has exactly one start transition
- The start transition is executed at process creation (do not call required interfaces there)
- The start transition
  - Sets the initial state
  - May execute initial actions (initialization of variables)

State / Nextstate



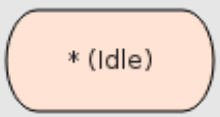
- Each state has a name
- In a given state, the process is expecting to receive messages

Shortcuts



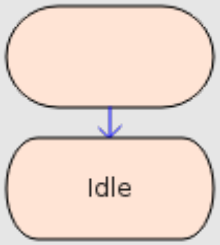
Multiple states

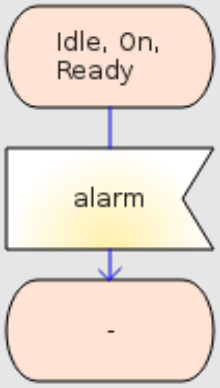
Any state



Any state except Idle

Shortcut

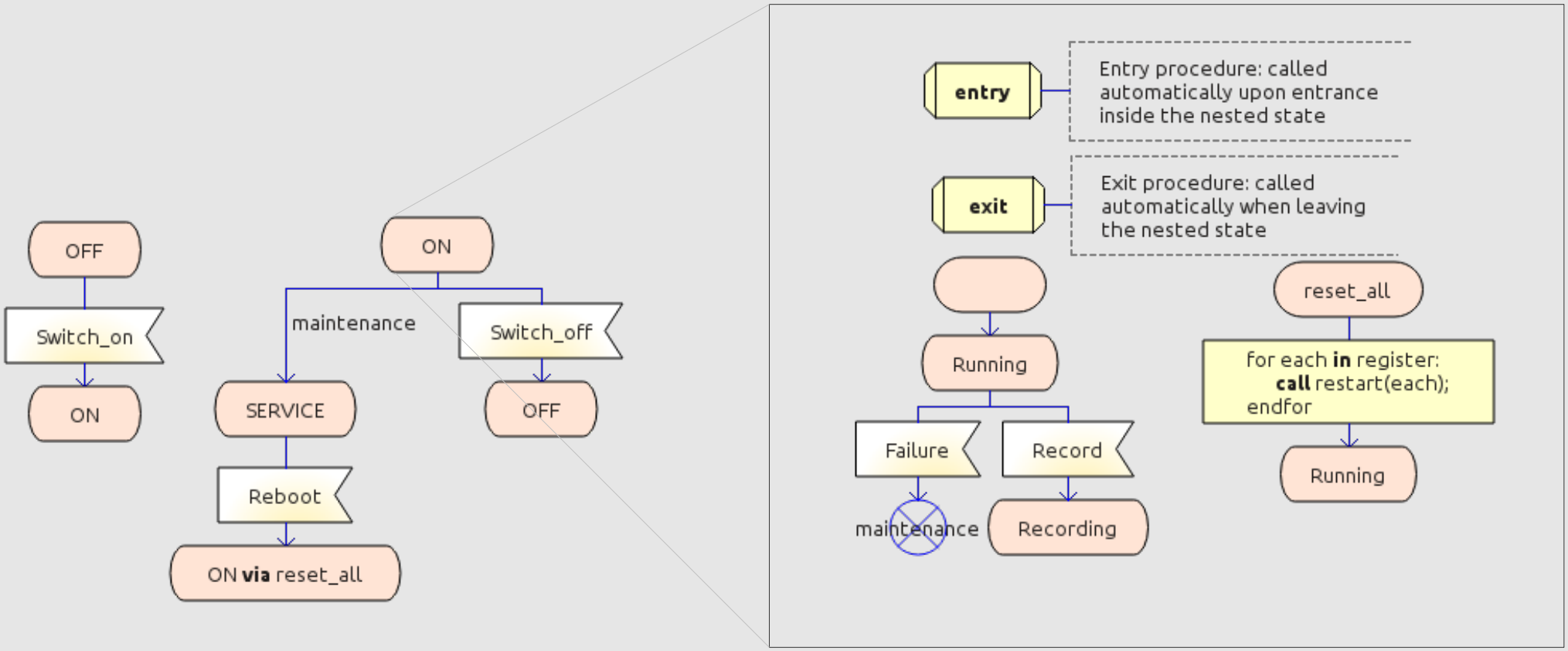




Most recent state

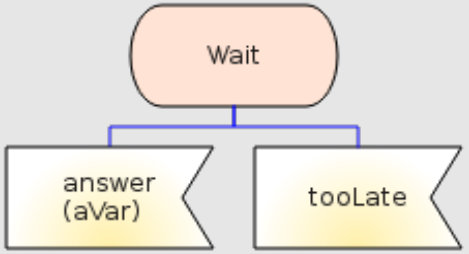
- Arrival state
- Unique
- Is the initial state of other transitions

Composite states



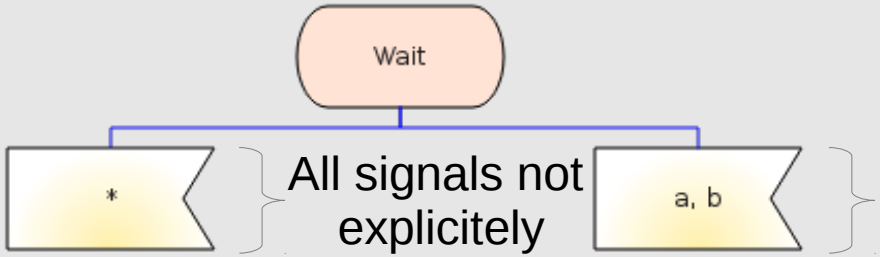
- Hierarchical state machines
- Entry and exit procedures
- Multiple entry and exit points

Input

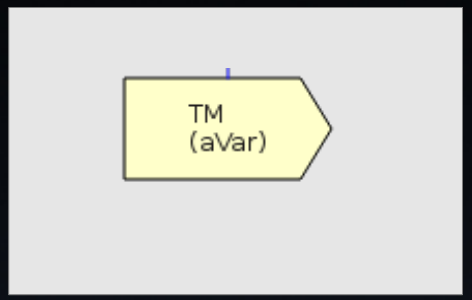


- Fires a transition : the transition is executed when the process consumes the signal
- In a given state, the process can expect several signals
- May have parameters (use variables to store their values)

Shortcuts

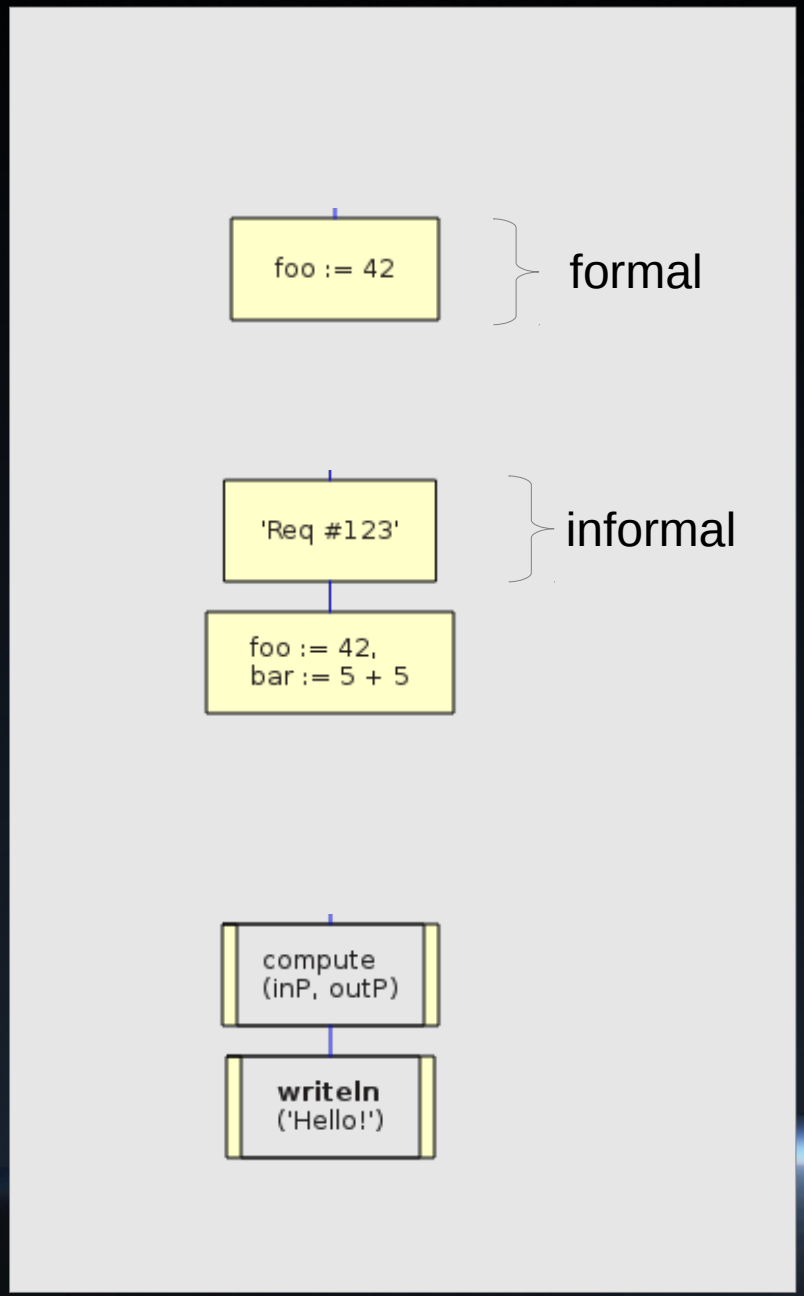


Output



- Transmission of a signal
  in TASTE terms : invocation of a sporadic required interface
- May have a parameter

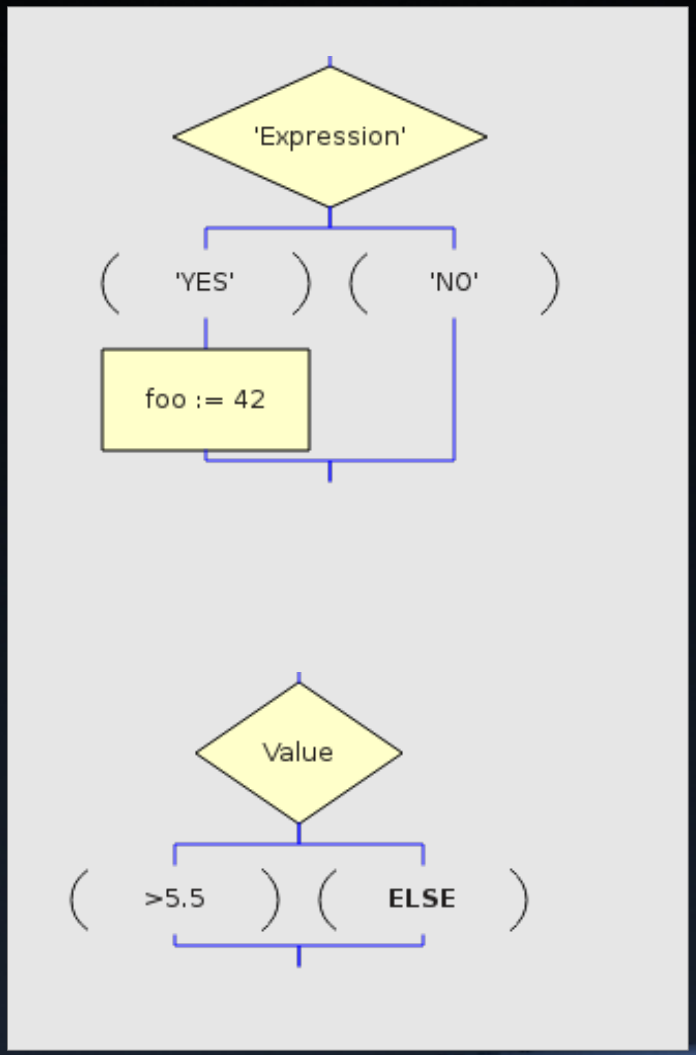
Task, Procedure call



- Elementary action of process transition
- Informal task
- Task setting a variable to a given value

- Call an external procedure *In TASTE terms, call a synchronous required interface (protected or unprotected)*
- Can have input and output parameters
- Writeln : built-in print function

Decision

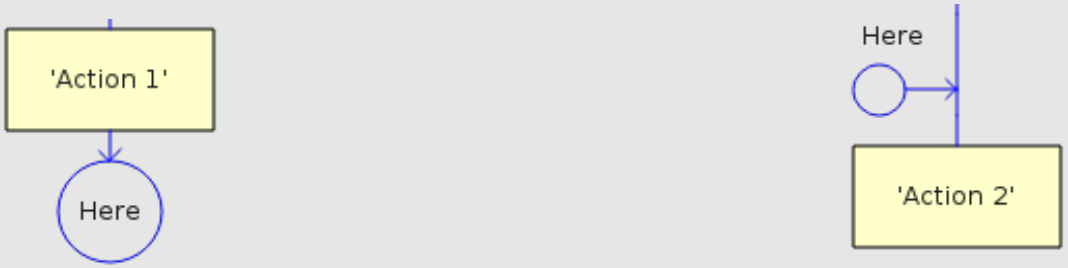


- Control structure
  To represent conditional action sequences
- A decision can have more than two answers

  - Multiple answers must be mutually exclusive

  -The last answer can be ELSE
- Useful to build loops

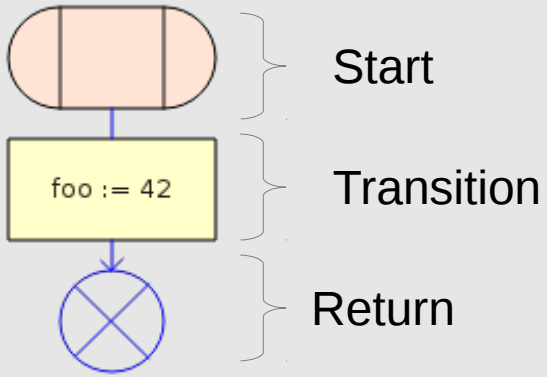
Labels and branches



- Allow rerouting
- Loop description
- « Don't repeat yourself » (DRY)

  But do not use to describe complex algorithms..

Procedures




```
-- A Local variable
DCL foo MyInteger;

-- Procedure interface
fpar
   in toto MyInteger,
   in/out tutu MyInteger;
```

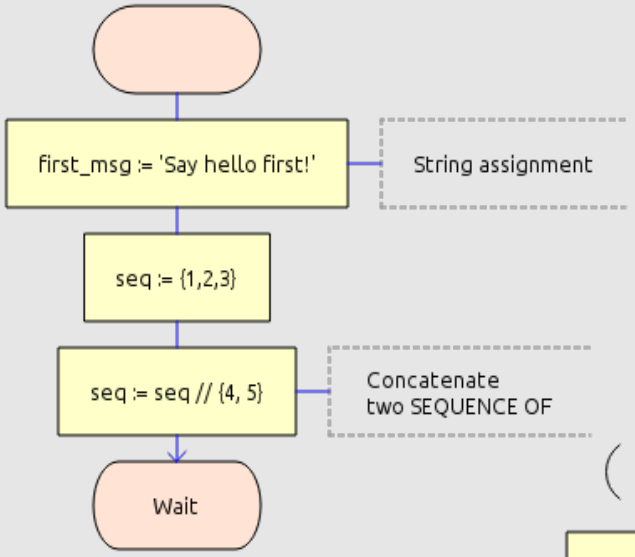
- Sequential sub-functions
- Can have parameters (in and in/out)
- Have visibility on the parent variables
- Same constructs as a process
- Local variables
- But no internal states

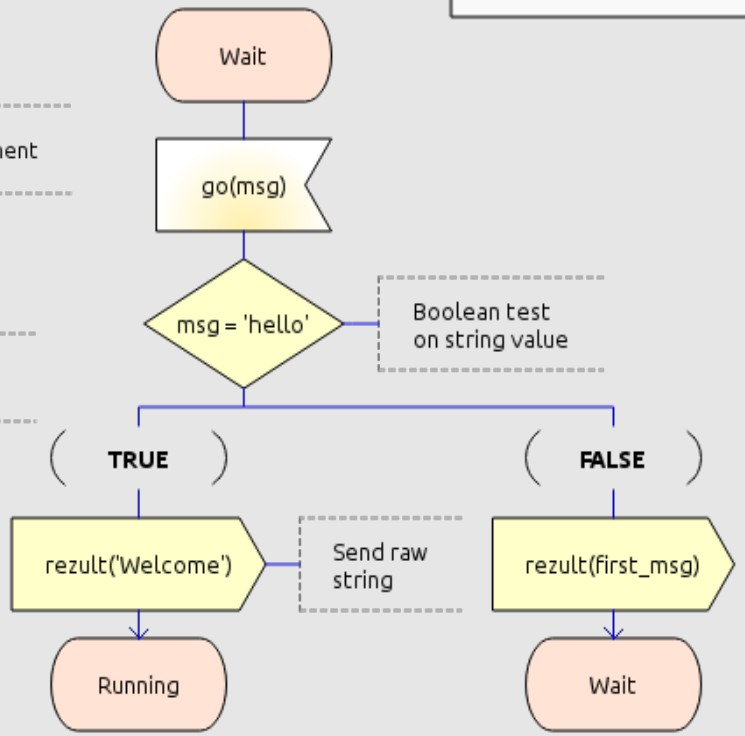
SDL and ASN.1

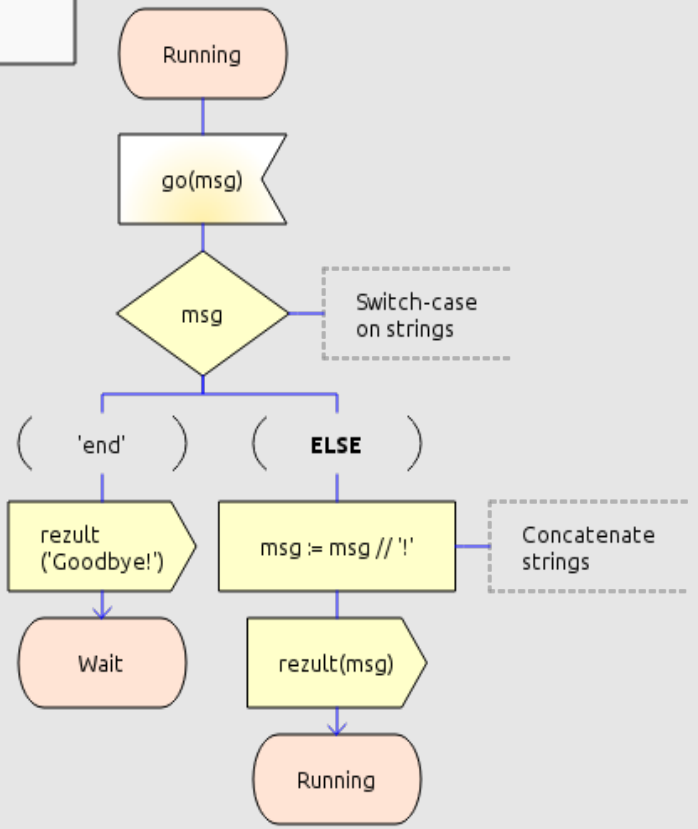
```
TASTE-Dataview DEFINITIONS ::=
BEGIN

MyInteger    ::= INTEGER (0..255)

MyOctStr     ::= OCTET STRING (SIZE (0..20))

SeqOf ::= SEQUENCE (SIZE(0..5)) OF MyInteger

END
~
"DataView.asn" 10L, 167C      1,1           Tout
```

- Declare variables of ASN.1 types
- Use strings and arrays

```
-- A Demo to test octet strings
-- using various symbols.

DCL first_msg, msg MyOctStr;

DCL seq SeqOf;
```







Quality criteria for state machines

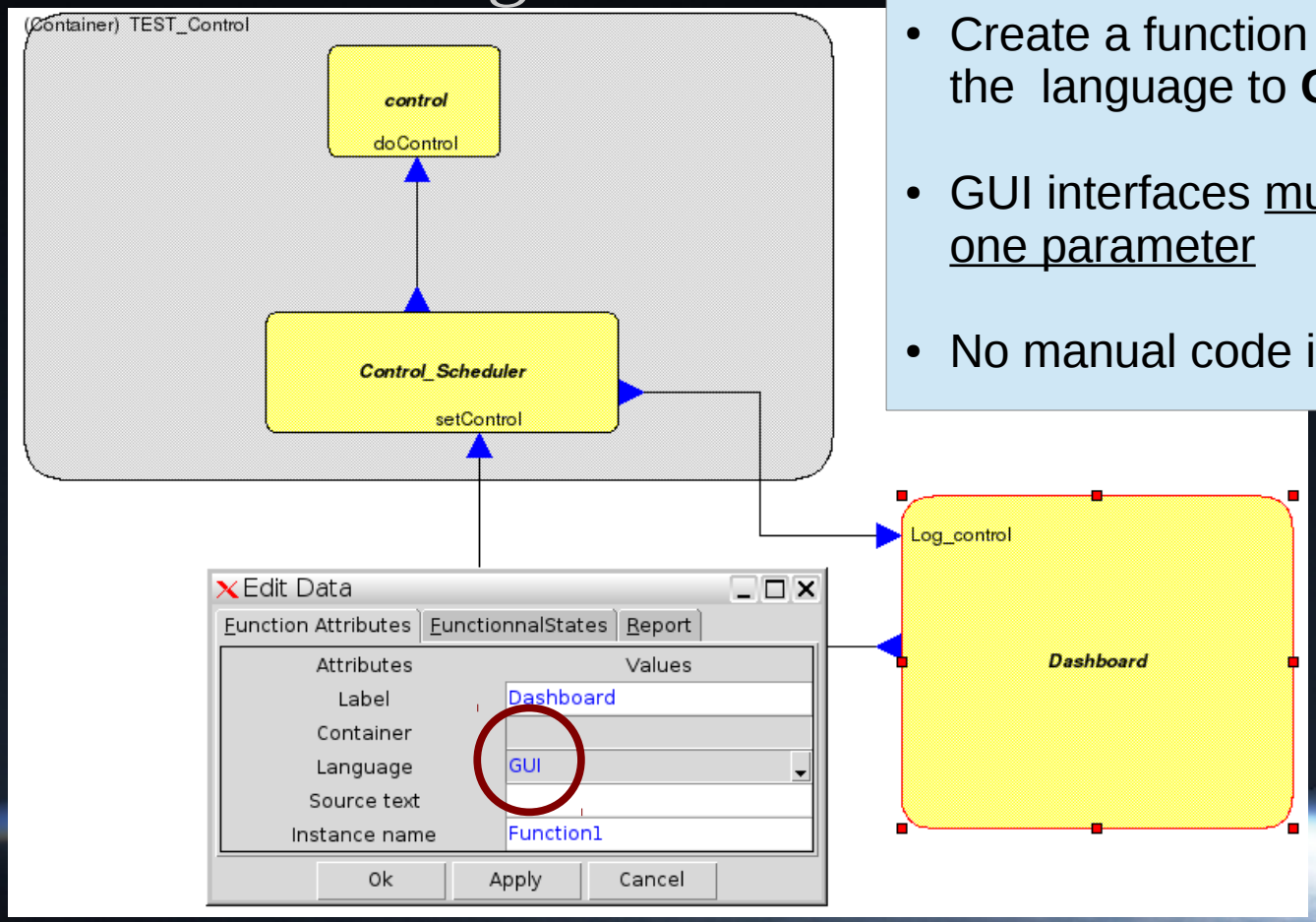
- State oriented
  - Use variables for storing data, not object states
- Complexity
  - Number of states
  - Number of transitions per state
  - Avoid decisions in waterfall
  - Minimum of data
- Graphical justification comments
- Use hyperlinks for better traceability

Summary

- SDL includes a complete data model
  - Declare and use variables within transition symbols
- Design is complete
  - Designers without expertise in programming languages can build complete executable models
  - TASTE allows communication with external code
- Best approach : model behaviour with SDL, algorithms with Simulink, and drivers with Ada or C

Graphical user interfaces

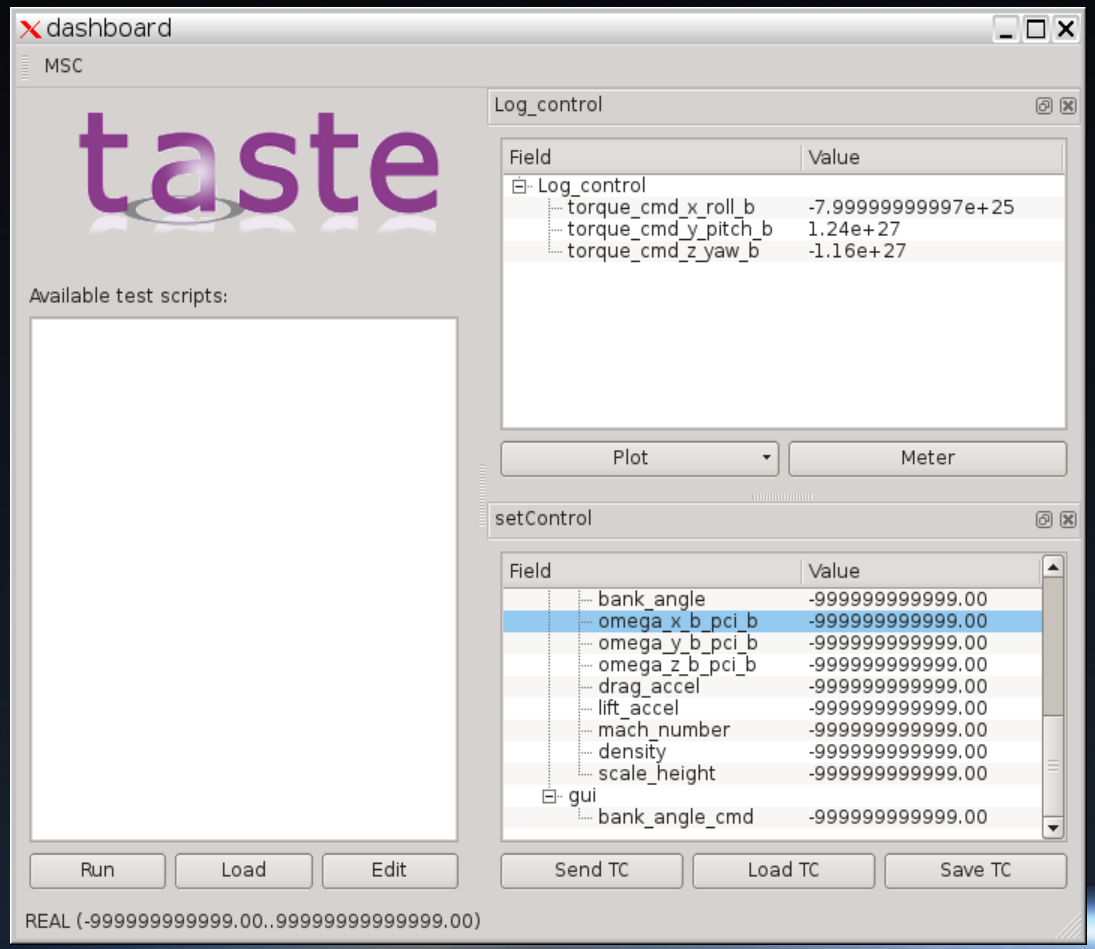
- Interactive execution
- Use for unit testing



- Edit the interface view :
  $ taste-edit-interface-view
- Create a function and set the language to **GUI**
- GUI interfaces must always hold one parameter
- No manual code is required

Result

- Creates an additional binary

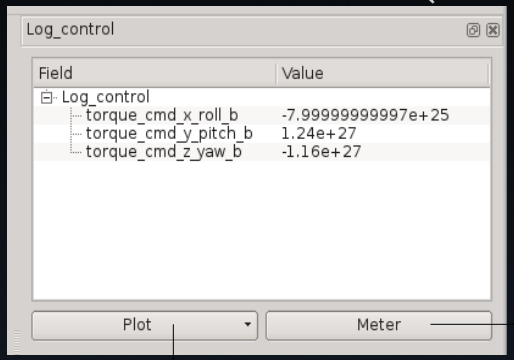


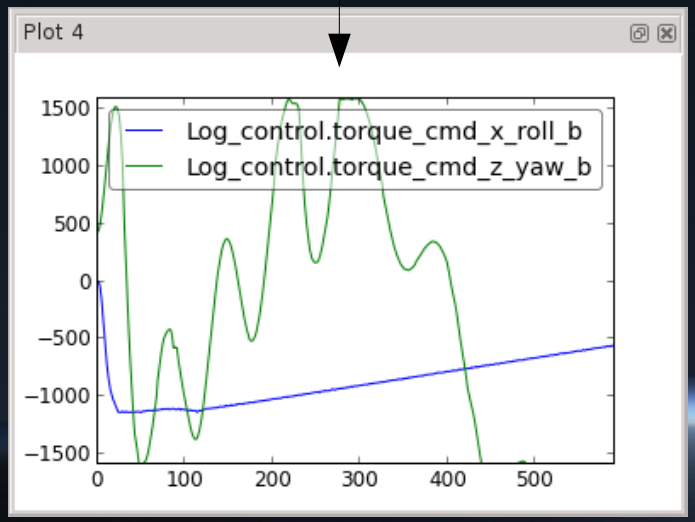
The GUI provided interfaces, data is updated each time the interface is called

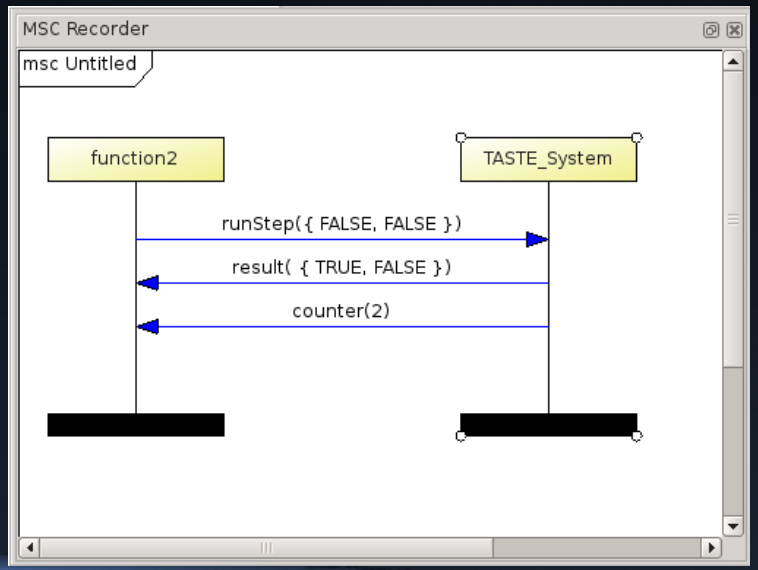
The GUI required interfaces, you can fill data and send the messages

Useful features

- Plot numerical data (in real-time)
- Record MSC (sequence diagrams)



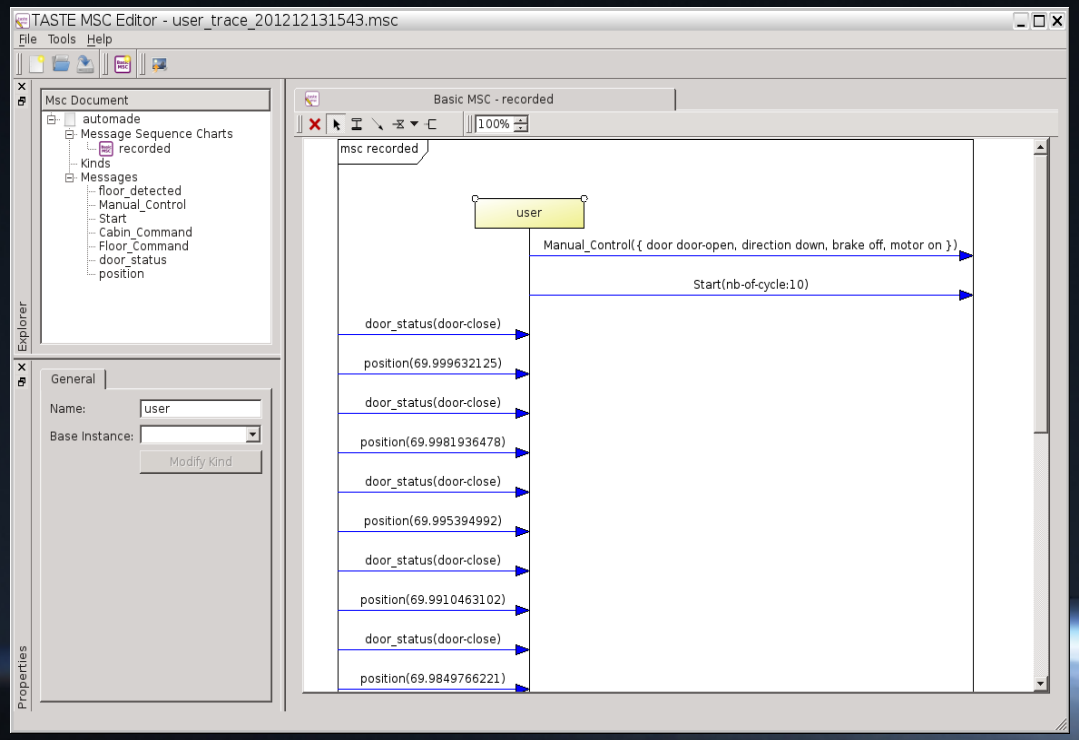




The built-in MSC editor

- Edit MSC, modify the recorded scenario
- Re-run the recorded scenario

  ⟶ Regression testing



- Describe the scenario you want to see (verify)
- Execute it against the running binary
- If message ordering or parameters are different than the expected scenario, an error will be raised

Ultimate testing power : Python

- MSC scenarii are translated to Python
- Edit one of these Python scripts to get a template, and write full-featured test suites

- Example of application : unit testing of a control law developed with Simulink

Python API (1)

- Write a scenario using Python decorators
- Parallel scenarii can run concurrently

```
@Scenario
def MyScenario(self):
    """ Run in parallel - send periodic messages """
    for i in xrange(10):
        self.sendMsg('runstep', '{ FALSE, TRUE }')
        time.sleep(1)
```

Python API (2)

- Send a message
- Wait for a specific message

```
expectMsg('Hello', '{ name *, age 35 }')   -> Must receive the "Hello" message with the parameter "age" having value 35. Name is not checked.
expectMsg('Hello', '*', ignoreOther=True)  -> Wait until it received "Hello", whatever the parameters
```

- Wait for any message

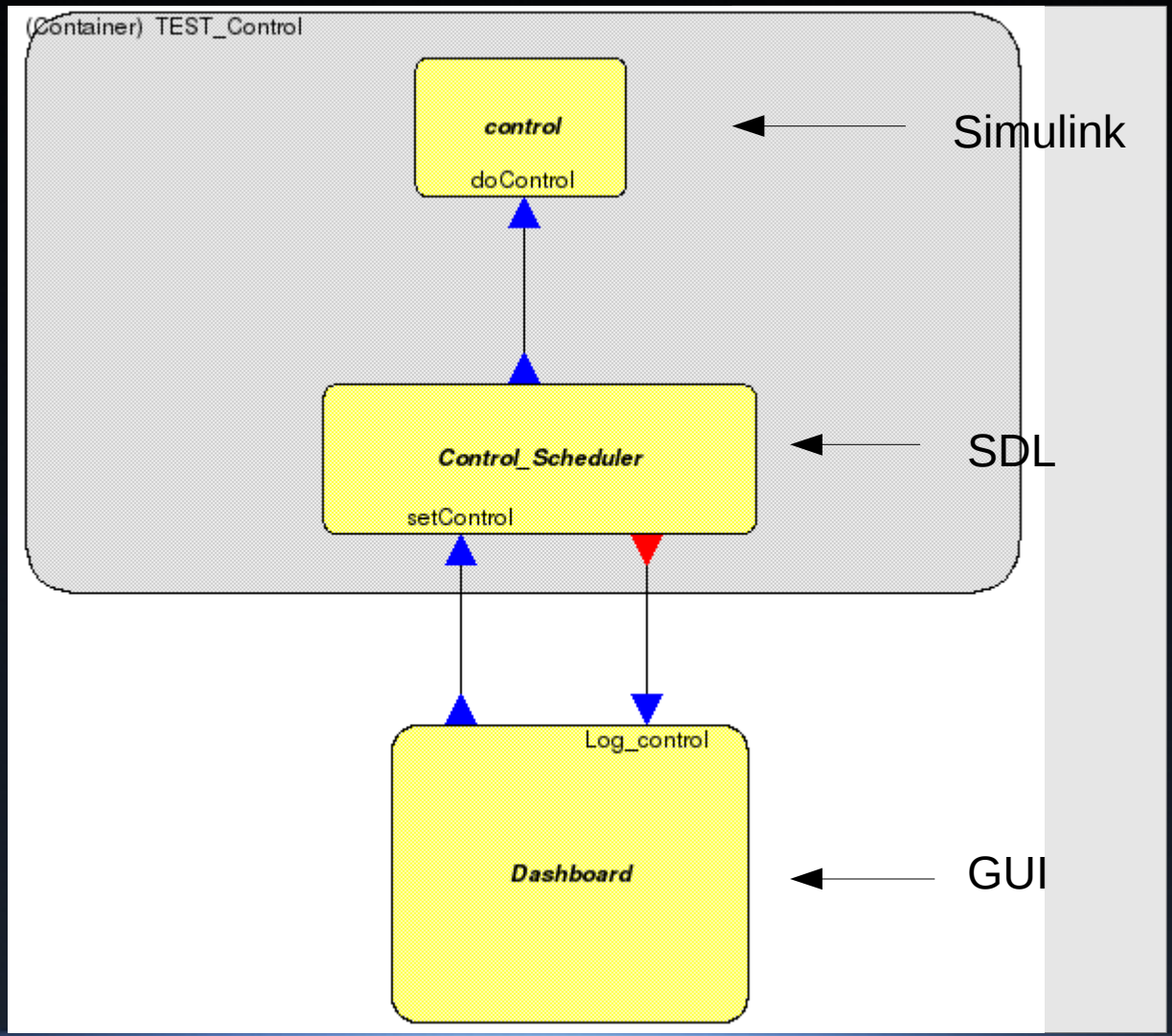
```
(msgId, val) = getNextMsg(timeout=10)
if msgId == 'Hello':
    print 'My name is', val.name.Get()  # Note the Python/ASN.1 API that allows to access to fields as if there were parameters
```

Case study : GNC Unit test

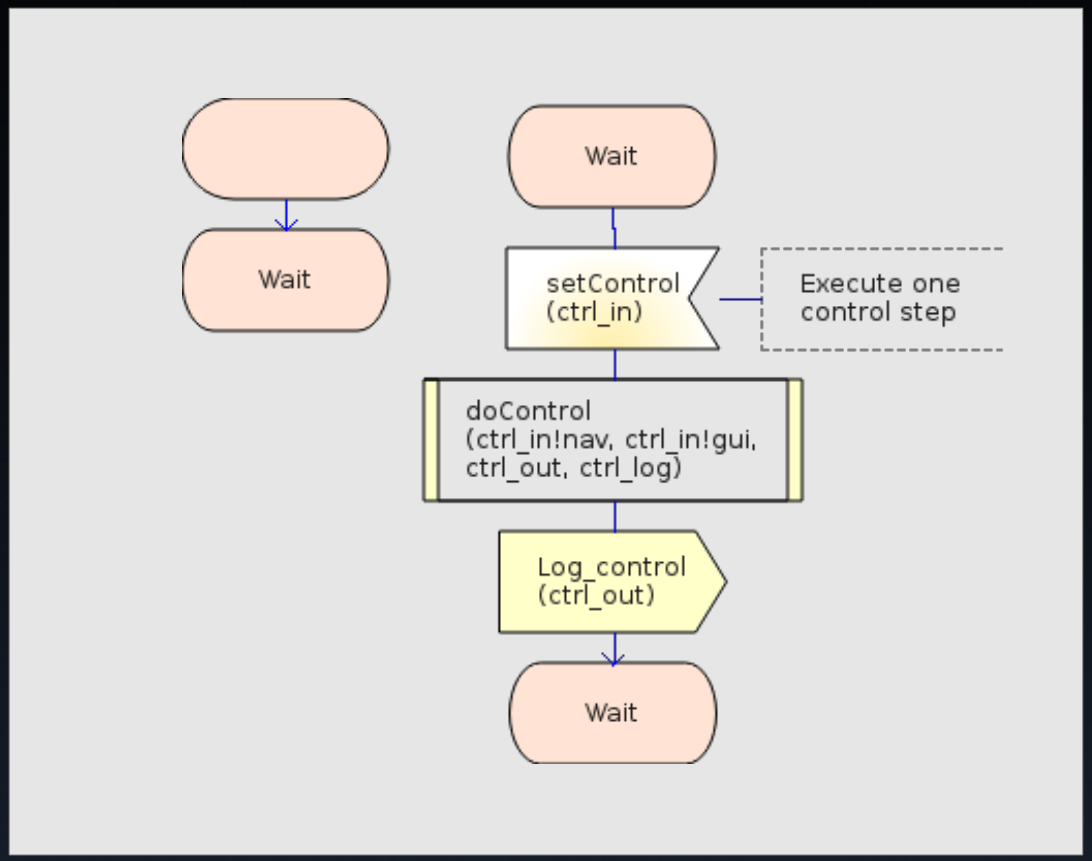
- We want to verify the Control part of a GNC
- Input : navigation and guidance, comes from a Simulink model (csv file with 3200 samples)
- Output : torques – we want to check the curves

- Interactive GUI is not adapted – needs automated processing. Plotting can be postponed

TASTE model (interface view)



Control_Scheduler SDL block




```
-- Declare your variables
DCL ctrl_in Control_input;
DCL ctrl_out Con_out;
DCL ctrl_log Log_con;
```

Test script

testScript.py + (~/work/GNC/ATPE_Ctrl/binary/binaries/dashboard-GUI) - GVIM1

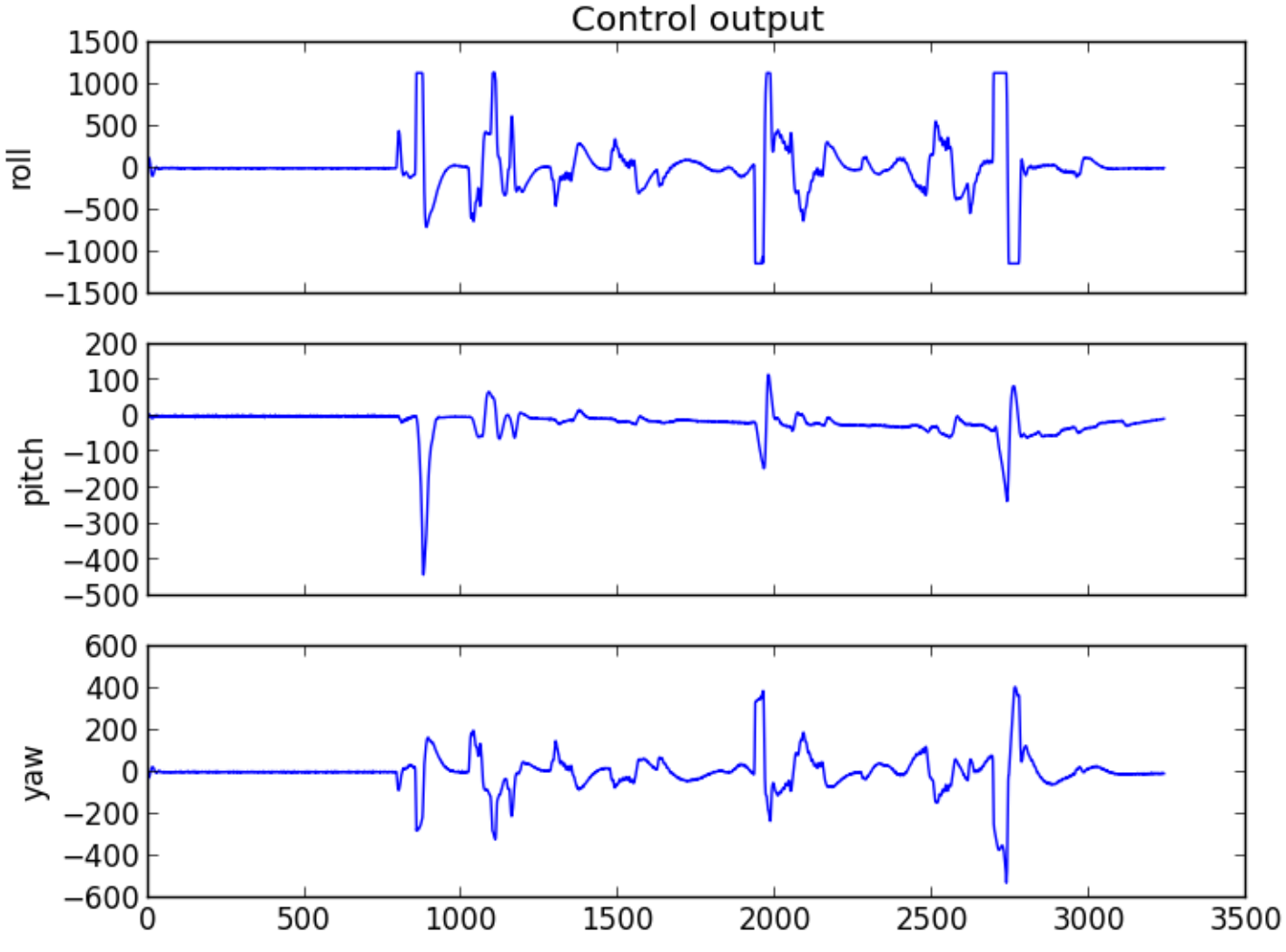
File Edit Tools Syntax Buffers Window Help

```
@Scenario
def Exercise_dashboard(self):
    '''dashboard processing'''
    roll, pitch, yaw = [], [], []
    # with open('ctrl_input.csv', 'rt') as inp:
    with open('ctrl_input.csv', 'rt') as inp:
        inp_lines = csv.reader(inp, delimiter=',')
        input_data = ('{{gui {{ bank-angle-cmd {l[22]} }},\
                nav {{ scale-height {l[21]},\
                    fp-speed {l[8]},\
                    omega-y-b-pci-b {l[15]}, \
                    mach-number {l[19]},\
                    density {l[20]},\
                    lift-accel {l[18]},\
                    fp-inclination-gc {l[9]},\
                    aoa {l[11]},\
                    bank-angle {l[13]},\
                    omega-x-b-pci-b {l[14]},\
                    aos {l[12]},\
                    fp-azimuth-gc {l[10]},\
                    drag-accel {l[17]},\
                    longitude {l[7]},\
                    velocity-pci-z {l[5]},\
                    omega-z-b-pci-b {l[16]},\
                    declination {l[6]},\
                    velocity-pci-y {l[4]},\
                    velocity-pci-x {l[3]},\
                    position-pci-z {l[2]},\
                    position-pci-y {l[1]},\
                    position-pci-x {l[0]} }} }}'.format(l=l) for l in inp_lines)
        for i in input_data:
            print 'Sending', i
            self.sendMsg('setControl', i, lineNo=0)
            (msgId, val) = self.getNextMsg(timeout = 1)
            if msgId and msgId == 'Log_control':
                roll.append(val.torque_cmd_x_roll_b.Get())
                pitch.append(val.torque_cmd_y_pitch_b.Get())
                yaw.append(val.torque_cmd_z_yaw_b.Get())
            else:
                print 'Unexpected message, quitting...'
                sys.exit(-1)

        # display plots
        f, axarr = plt.subplots(3, sharex=True)
        axarr[0].plot(roll)
        axarr[0].set_ylabel('roll')
        axarr[1].plot(pitch)
        axarr[1].set_ylabel('pitch')
        axarr[2].plot(yaw)
        axarr[2].set_ylabel('yaw')
        axarr[0].set_title('Control output')
        plt.show()
    return 0

def runScenario(udpController=None, callback=None):
```

59,0-1 40%

- Open CSV file and map columns to the input vector
- Send each input, wait for the output, and store it
- Plot the output using Matplotlib

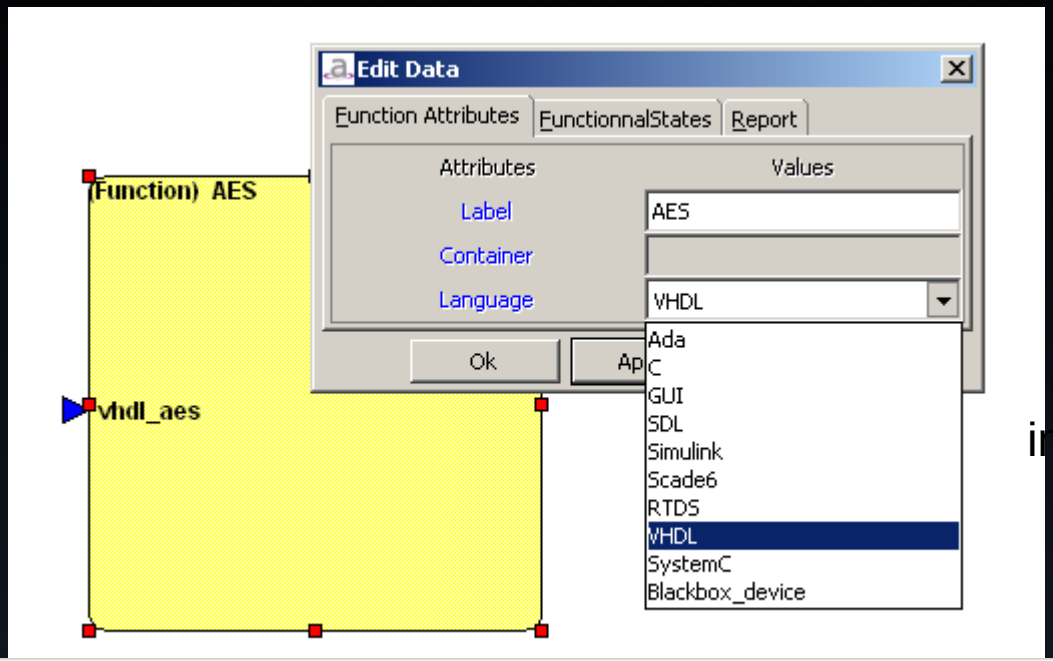
Result


Control output
roll
pitch
yaw
1500
1000
500
0
−500
−1000
−1500
200
100
0
−100
−200
−300
−400
−500
600
400
200
0
−200
−400
−600
0
500
1000
1500
2000
2500
3000
3500

More TASTE features

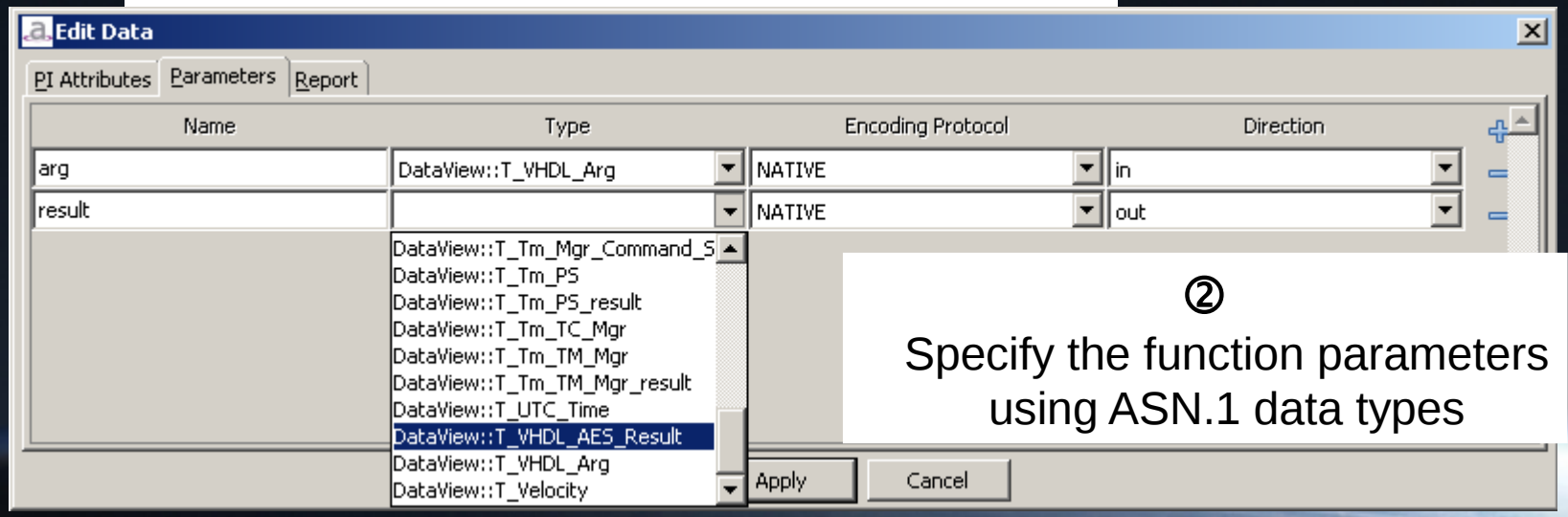
- Support for FPGA development
- Import/Export of components
- PeekPoke to tweak internal data at runtime
- Blackbox devices to write drivers
- MAST, Cheddar and Marzhin for scheduling analysis
- Coverage, Profiling
- Windows GUI
- SMP2 import/export mechanism to work with satellite simulators (Simsat, Basiles, Eurosim)

FPGA support



①

Specify a function in the interface view and set the implementation language to VHDL



②

Specify the function parameters using ASN.1 data types

VHDL Generated interface

```vhdl
library ieee;
use ieee.std_logic_1164.all;

use work.config.all;

        entity vhdl_aes is
        port (
            arg_choiceIdx : in std_logic_vector(7 downto 0);
            arg_t_vhdl_aes_arg_set_key_t_arg_key_length : in std_logic_vector(7 downto 0);
            arg_t_vhdl_aes_arg_set_key_t_arg_key_content: in octStr_32;
            arg_t_vhdl_aes_arg_encrypt_t_arg_encrypt_direction : in std_logic_vector(7 downto 0);
            arg_t_vhdl_aes_arg_encrypt_t_arg_encrypt_in: in octStr_16;
            result_choiceIdx : out std_logic_vector(7 downto 0);
            result_t_vhdl_aes_result_status : out std_logic;
            result_t_vhdl_aes_result_out: out octStr_16;
            start_vhdl_aes  : in  std_logic;
            finish_vhdl_aes : out std_logic;
            rst_vhdl_aes    : in std_logic;
            clk_vhdl_aes    : in  std_logic
        );
        end vhdl_aes;
```

Code skeleton to be filled by the user

```
architecture archivhdl_aes of vhdl_aes is

begin
        process(clk_vhdl_aes,rst_vhdl_aes)
                variable run : std_logic;
        begin
                if rst_vhdl_aes='0' then -- Asynchronous reset

                        finish_vhdl_aes <= '0';
                        -- write your resets here
                        run := '1';

                elsif clk_vhdl_aes'event and clk_vhdl_aes='1' then

                        if start_vhdl_aes = '0' then
                                finish_vhdl_aes <= '0';
                                run := '1';
                        elsif run = '1' then

                                -- write your logic to compute outputs from inputs here
                                -- and when your results are ready, set...
                                --
                                -- run := '0';
                                -- finish_vhdl_aes <= '1';
                        end if;
                end if;
        end process;
end archivhdl_aes;
```

Glue code generated by TASTE

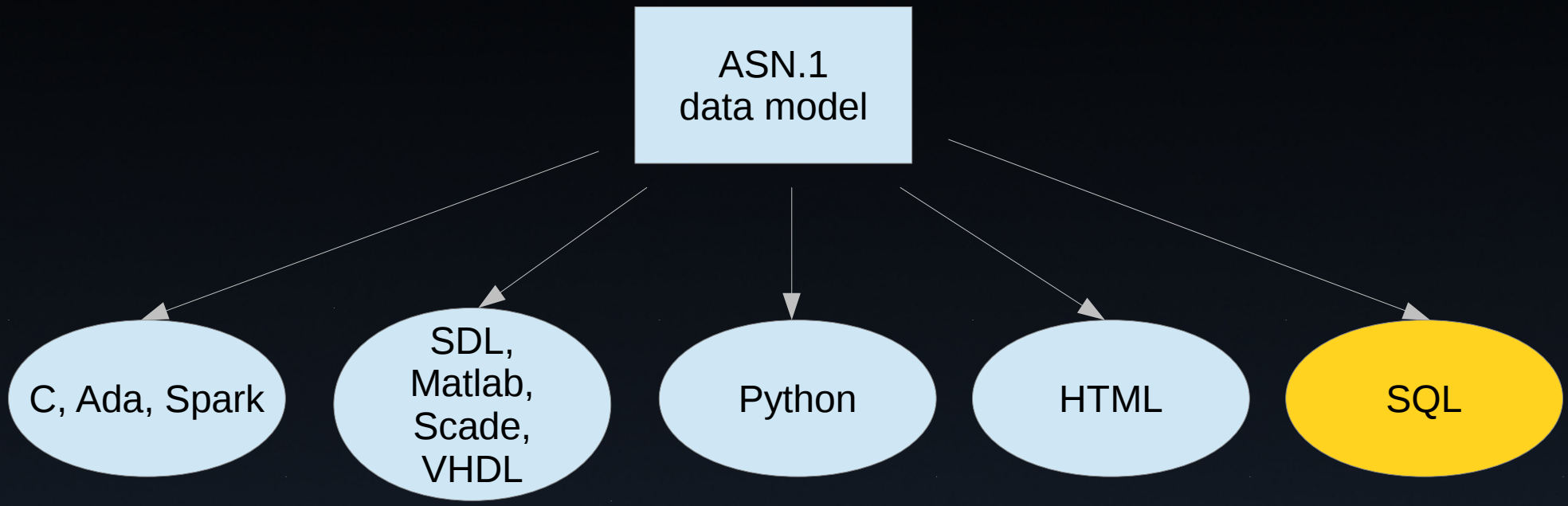
- Glue in VHDL (IP cores to read/write data on the PCI bus)
- Glue code on the Leon side

```
if (var_T_VHDL_Arg.kind == t_vhdl_aes_arg_set_key_PRESENT) {
    unsigned tmp = 1;
    ESAWriteRegister(BASE_ADDR + 0x4, tmp);
    {
        unsigned tmp = var_T_VHDL_Arg.u.t_vhdl_aes_arg_set_key.t_arg_key_length;
        ESAWriteRegister(BASE_ADDR + 0x8, tmp);
    }
    {
        unsigned tmp = 0;
        tmp |= ((unsigned)var_T_VHDL_Arg.u.t_vhdl_aes_arg_set_key.t_arg_key_content.arr[0]) << 0;
        tmp |= ((unsigned)var_T_VHDL_Arg.u.t_vhdl_aes_arg_set_key.t_arg_key_content.arr[1]) << 8;
        tmp |= ((unsigned)var_T_VHDL_Arg.u.t_vhdl_aes_arg_set_key.t_arg_key_content.arr[2]) << 16;
        tmp |= ((unsigned)var_T_VHDL_Arg.u.t_vhdl_aes_arg_set_key.t_arg_key_content.arr[3]) << 24;
        ESAWriteRegister(BASE_ADDR + 0xc + 0, tmp);
    }
```

Import-Export components

- Right click in the Interface view to import or export components
- Example : the PeekPoke component
- Used to monitor runtime data (e.g. Simulink tuneable parameters) without user code modification.
- Allow to modify data in memory at runtime
- Useful to tune algorithms

ASN.1 to SQL / Working with databases



TASTE relies on ASN.1 to ensure consistency of data at each level of the process :
Engineering, processing, testing, documentation, communication, data storage and retrieval.

ASN.1 to SQL magic

- Use the same ASN.1 model to create SQL schemas ⟶ keep consistency (one SQL table per ASN.1 data type is created by the toolchain, automatically)
- Use case : telecommand/telemetry storage
  - Describe TM/TC data format in ASN.1 and ACN
  - Use C/Ada binary encoder/decoders in flight code
  - Use ICD generator to document format at binary level
  - Pick TC/Store TM in the SQL database for post-processing – field format is correct by construction
- Very flexible : using SQLAlchemy to be compatible with Oracle, SQLite, PostgreSQL...
- Python interface

A simple API

```
MyInt ::= INTEGER (0..20)
```

```
# Can work with any DB. Here is an example with PostgreSQL
engine = create_engine(
    'postgresql+psycopg2://taste:tastedb@localhost/test', echo=False)


# Create data using the ASN.1 Python API
a = MyInt()
a.Set(5)


# Add the value to the SQL table called MyInt
aa1 = MyInt_SQL(a)
aid1 = aa1.save(session)
```

A simple API – Retrieve data

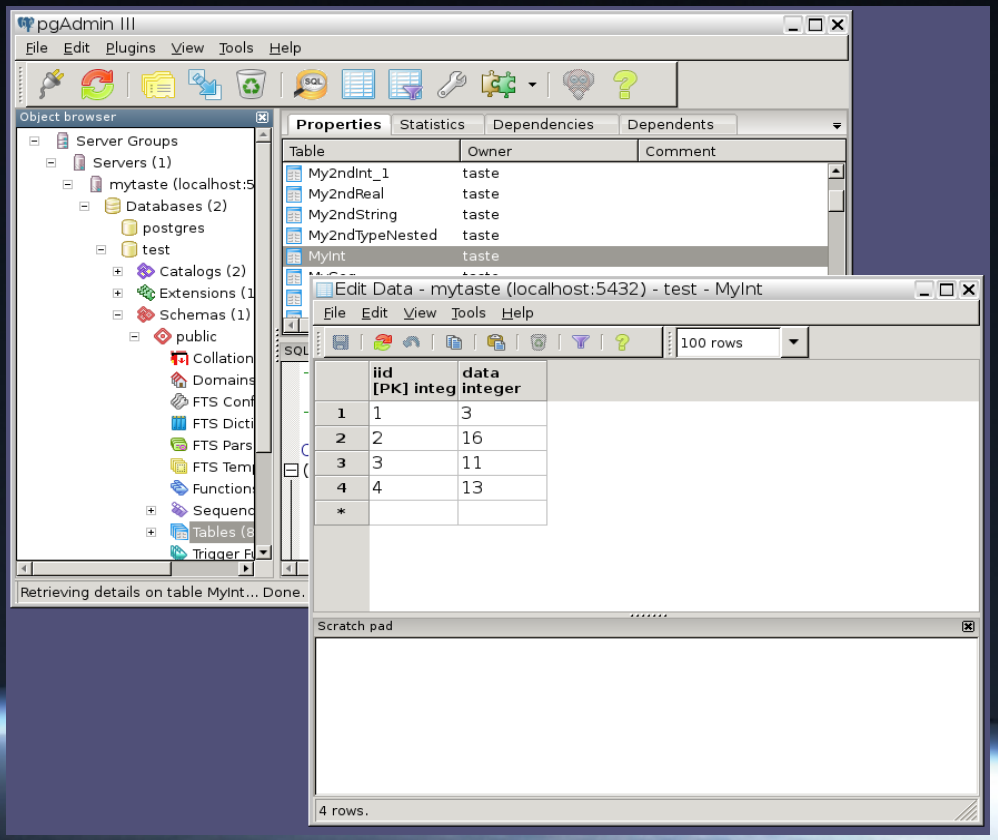
```
# Data is retrieved using SQL queries, or SQLAlchemy API

# Retrieve ALL records in the MyInt table
all_values = self.session.query(MyInt_SQL)

for record in all_values:
    # The magic : data is transparently converted back to ASN.1
    print record.asn1.Get()
```

Query data with the full power of databases. It will be converted automatically to ASN.1 structures.

Use case :
Query all TC with type=XX and subtype=YY (1 line of code)
Select the ones you are interested in
Encode them with ASN.1/ACN to a PUS packet (1 line of code)
Send them to the satellite (1 line of code)

Check the results

- Demo of the complete features in `/home/assert/tool-src/DMT/tests-sqlalchemy`
- Run `make` (password for the db is *tastedb*)
- Run pgadmin3



Ecosystem

- Web entry point : http://taste.tuxfamily.org
- User (public) and developpers (private) mailing lists
  → **Register to the user mailing list** !
- Bug track system
- SVN repository with all sources (hosted at ESA)
  - Stable and trunk branches
- Nightly build and regression testing