opengeode

A tiny free, open-source state-machine editor and code generator, based on the SDL and ASN.1 languages.

Maxime Perrotin
Software Engineering division - ESTEC
TEC-SWE

www.esa.int

European Space Agency

Introduction - overview

- How to describe the behaviour of a system ?
-
- System engineers define their systems modes and transitions with « state machines »
- State machines have simple concepts and solid foundations – they hide implementation details and allow to focus first on the dynamics
- Unfortunately most programming languages (like C or Ada) do not have a grammar that natively handle these concepts
- …Except **SDL**, a 25+ years-old language that is dedicated to state machines, and widely used in telecom applications – why not trying it in space ?

State machines are useful...really ?

- Embedded systems react to their environment

```
void handle_event(Event evt)
{
    if (evt == unit_failure) bus_power_cycle() ;
}
```

- Spot the bug ?
- There is nothing to prevent the continuous power cycle....

Note : this example was inspired by the description of the State design pattern found at http://gameprogrammingpatterns.com/state.html

State machines are useful

- A simple fix : add a flag

```
void handle_event(Event evt)
{
    if (evt == unit_failure) {
            if (!power_cycling) {
                power_cycling = true;
                // Power cycle...
            }
    }
}
```

- Next, we want to calibrate some sensors...

State machines are useful

```
void handle_event(Event evt)
{
    if (evt == unit_failure) {
        // Power cycle if not power cycling...
    }
    else if (evt == calibrate) {
        if (!power_cycling) start_calibration() ;
    }
    else if (evt == calibration_end) {
        tm_up_and_running() ;
    }
}
```

- Spot the bug this time ?
  - Start calibration, receive a failure, end calibration..
  - The system will report it is up and running while in fact power cycling.
- Time for another flag !

State machines are useful

```
void handle_event(Event evt)
{
    if (evt == unit_failure) {
        if (!power_cycling && !calibrating)
        // Power cycle...
    }
    else if (evt == calibrate) {
        if (!power_cycling) { calibrating = true; ... }
    }
    else if (evt == calibration_end) {
        if(calibrating) { calibrating = false ; ...}
    }
}
```

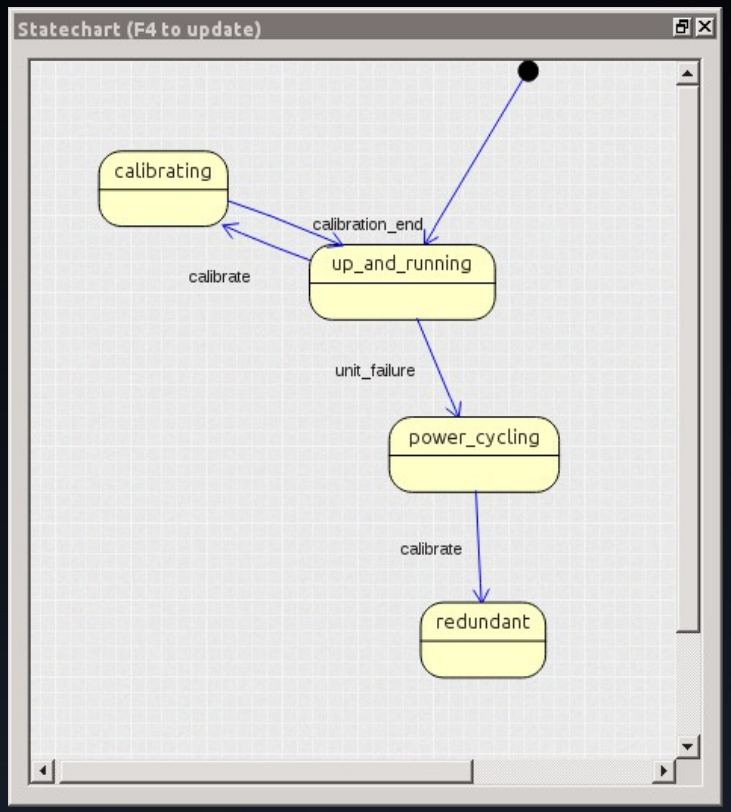
- Next, it would be nice to switch to redundant if a calibration is requested while power cycling

State machines are useful

```
void handle_event(Event evt)
{
    if (evt == unit_failure) {
        if (!power_cycling && !calibrating)
        // Power cycle...
    }
    else if (evt == calibrate) {
        if (!power_cycling) { calibrating = true; ... }
        else {
            power_cycling = false ;
            switch_on_redundant() ;
        }
    }
    else if (evt == calibration_end) {
        if(calibrating) { calibrating = false ; ...}
    }
}
```

- Bug hunting time again. Find it ?
- We check that we can't power cycle while calibrating but not while switching on redundant. Yet another flag....
- Etc, etc..
- Something is clearly wrong with this approach : every time we touch this handful of code, we break something.

State machines are useful...really !

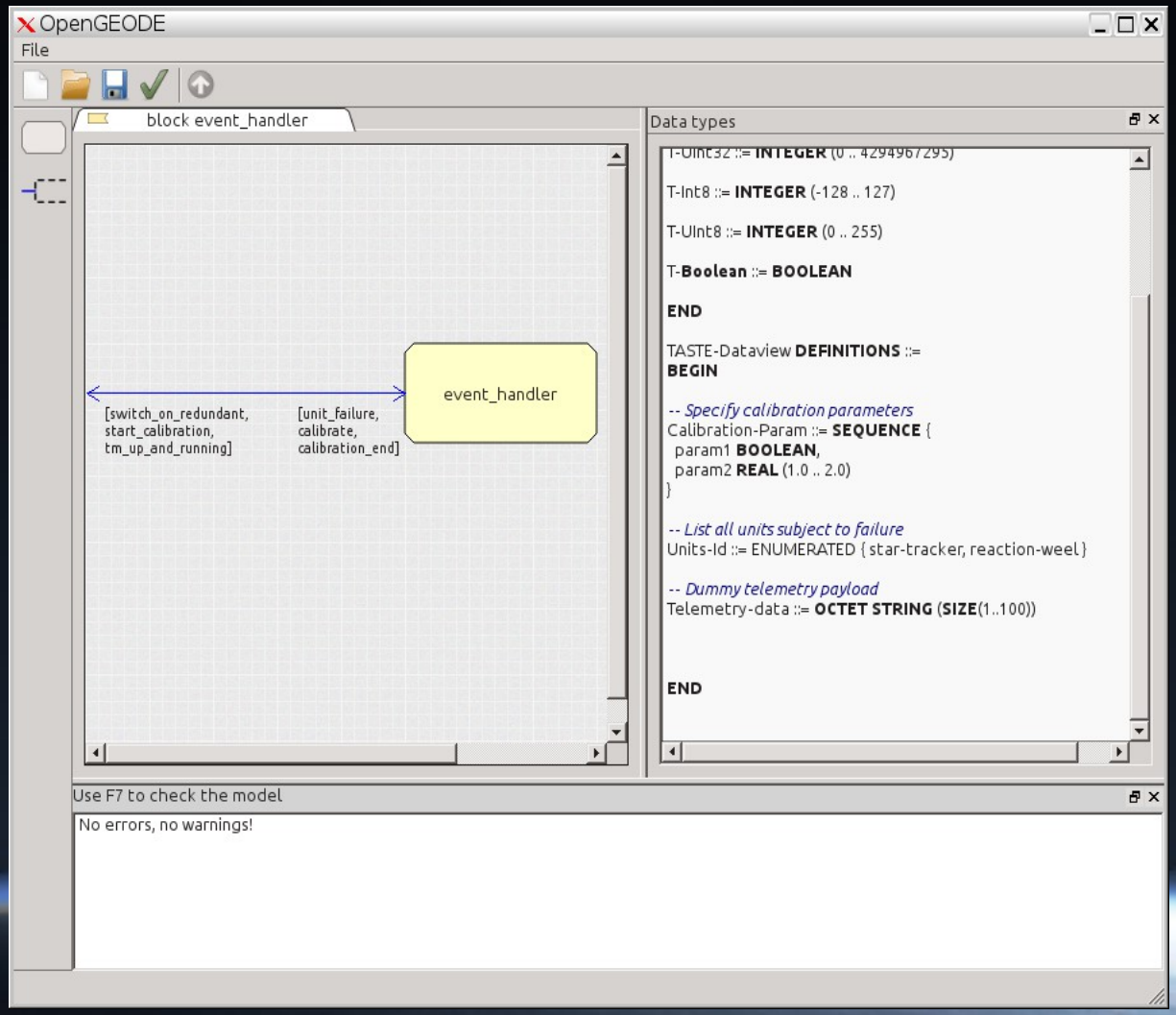


OpenGEODE

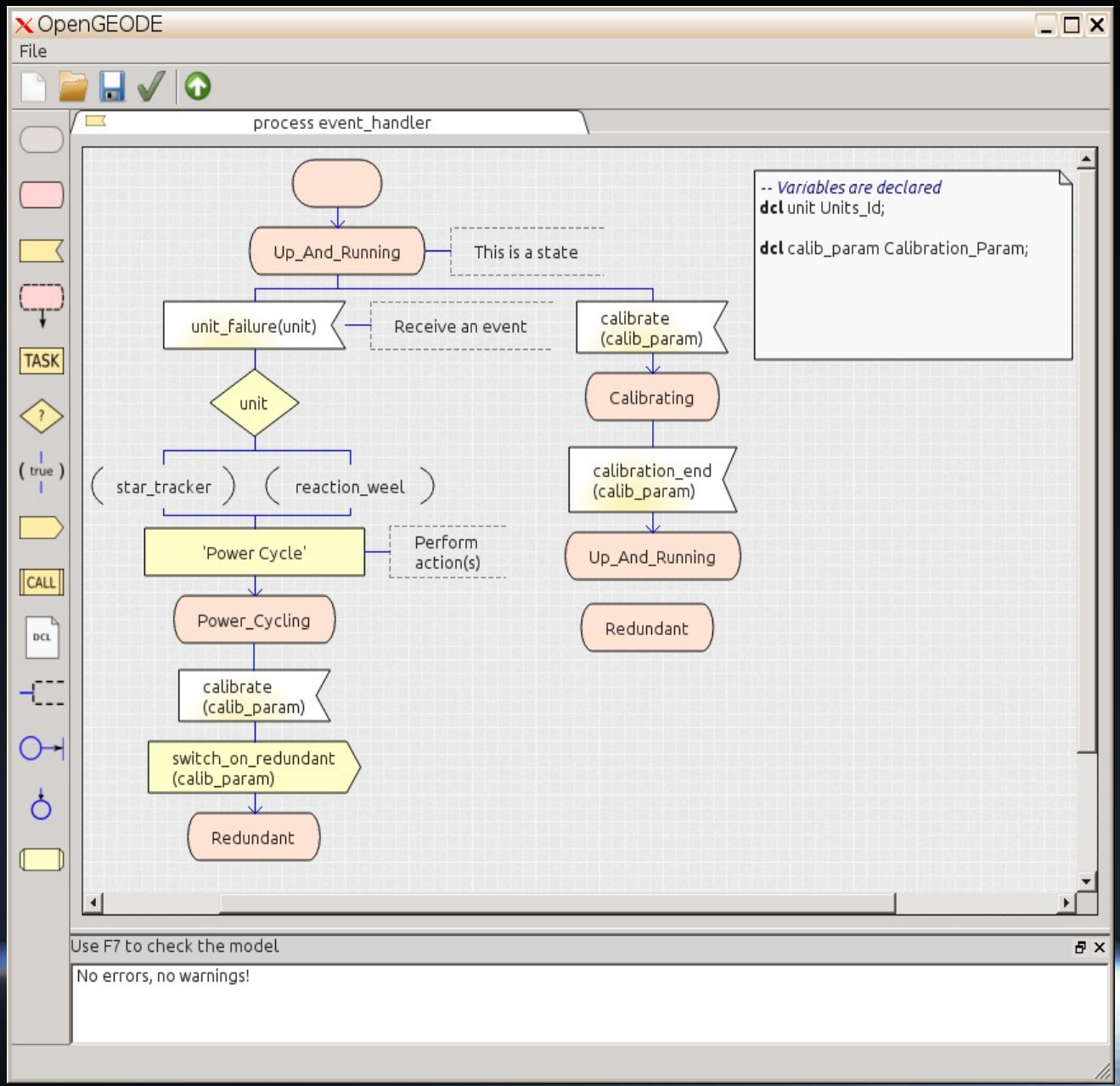
- OpenGEODE graphical editor and compiler based on the SDL language
- It is free and open-source, small and easy to maintain
- Development is active and 3rd party support is available
- The tool is a prototype and does not aim at competing with full-featured, commercial options (RTDS)
- The target is restricted to embedded/safe systems – using ASN.1, generating Ada code, ... with a special focus put on providing a great user experience

OpenGEODE

- Everything is formally specified : interface, data, and behaviour.



OpenGEODE


OpenGEODE
File
process event_handler
-- Variables are declared
dcl unit Units_Id;
dcl calib_param Calibration_Param;
Up_And_Running
This is a state
unit_failure(unit)
Receive an event
calibrate (calib_param)
unit
Calibrating
star_tracker
reaction_weel
calibration_end (calib_param)
'Power Cycle'
Perform action(s)
Up_And_Running
Power_Cycling
Redundant
calibrate (calib_param)
switch_on_redundant (calib_param)
Redundant
TASK
CALL
DCL
Use F7 to check the model
No errors, no warnings!

OpenGEODE

- Graphical and textual notations are equivalent
- The save format is the textual notation
- The syntax is simple and the parser is available
- Writing a backend is easy
- Two backends exist : Ada and LLVM code generators

```
PROCESS event_handler;
   -- Variables are declared
   dcl unit Units_Id;

   dcl calib_param Calibration_Param;

   START;
      NEXTSTATE Up_And_Running ;
   STATE Redundant;
   ENDSTATE;

   STATE Calibrating;
      INPUT calibration_end(calib_param);
         NEXTSTATE Up_And_Running;
   ENDSTATE;

   STATE Power_Cycling;
      INPUT calibrate(calib_param);
         OUTPUT switch_on_redundant(calib_param);
         NEXTSTATE Redundant;
   ENDSTATE;

   STATE Up_And_Running  COMMENT 'This is a state';
      INPUT unit_failure(unit) COMMENT 'Receive an event';
         DECISION unit;
               (reaction_weel):
               (star_tracker):
         ENDDECISION;
         TASK 'Power Cycle'  COMMENT 'Perform action(s)';
         NEXTSTATE Power_Cycling;
      INPUT calibrate(calib_param);
         NEXTSTATE Calibrating;
   ENDSTATE;
ENDPROCESS event_handler;
```

SDL and ASN.1 – two complementary languages

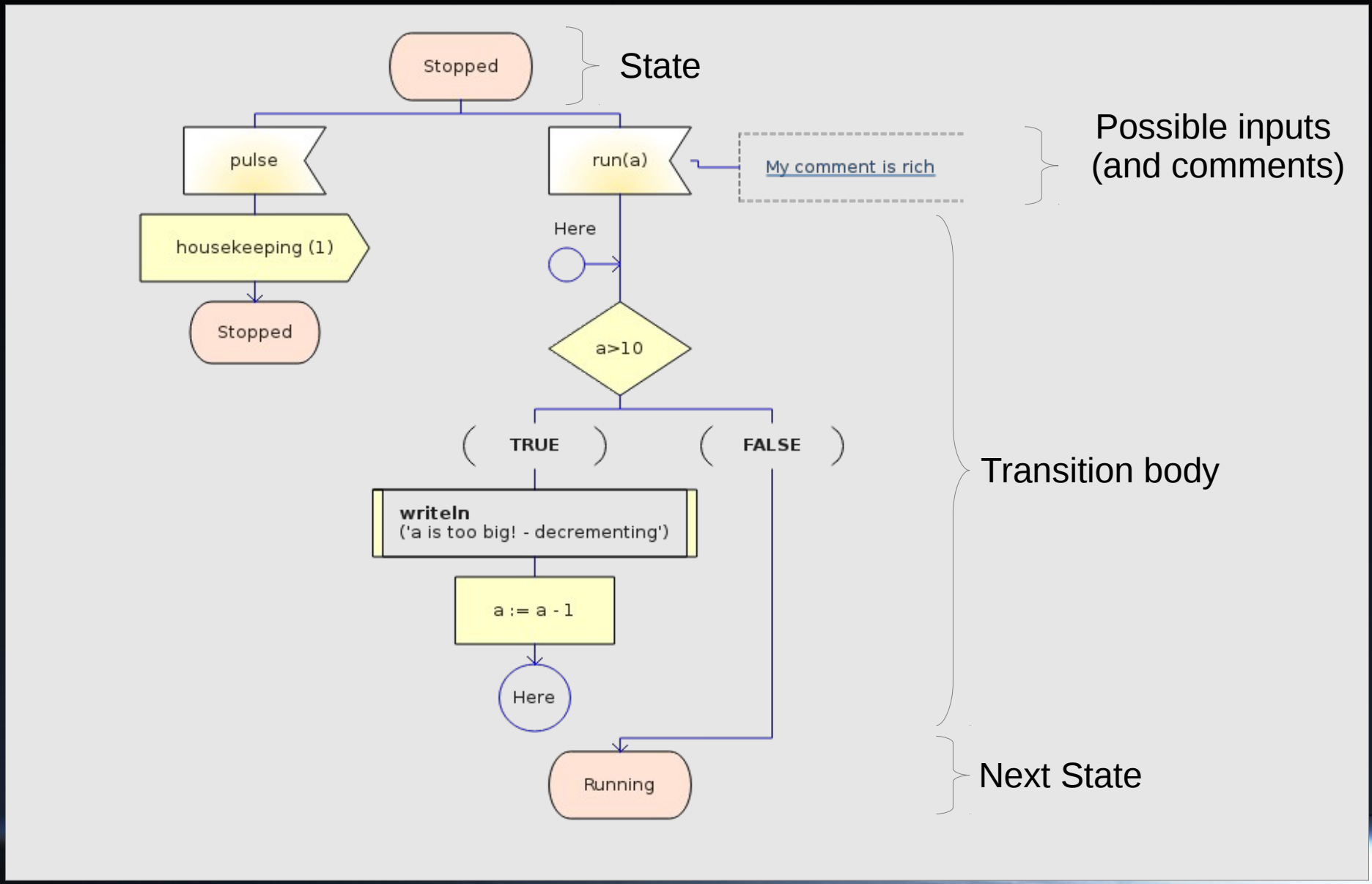
- **SDL : Specification and Description Language (ITU-T standard)**
- **ASN.1 : Abstract Syntax Notation One**

ASN.1

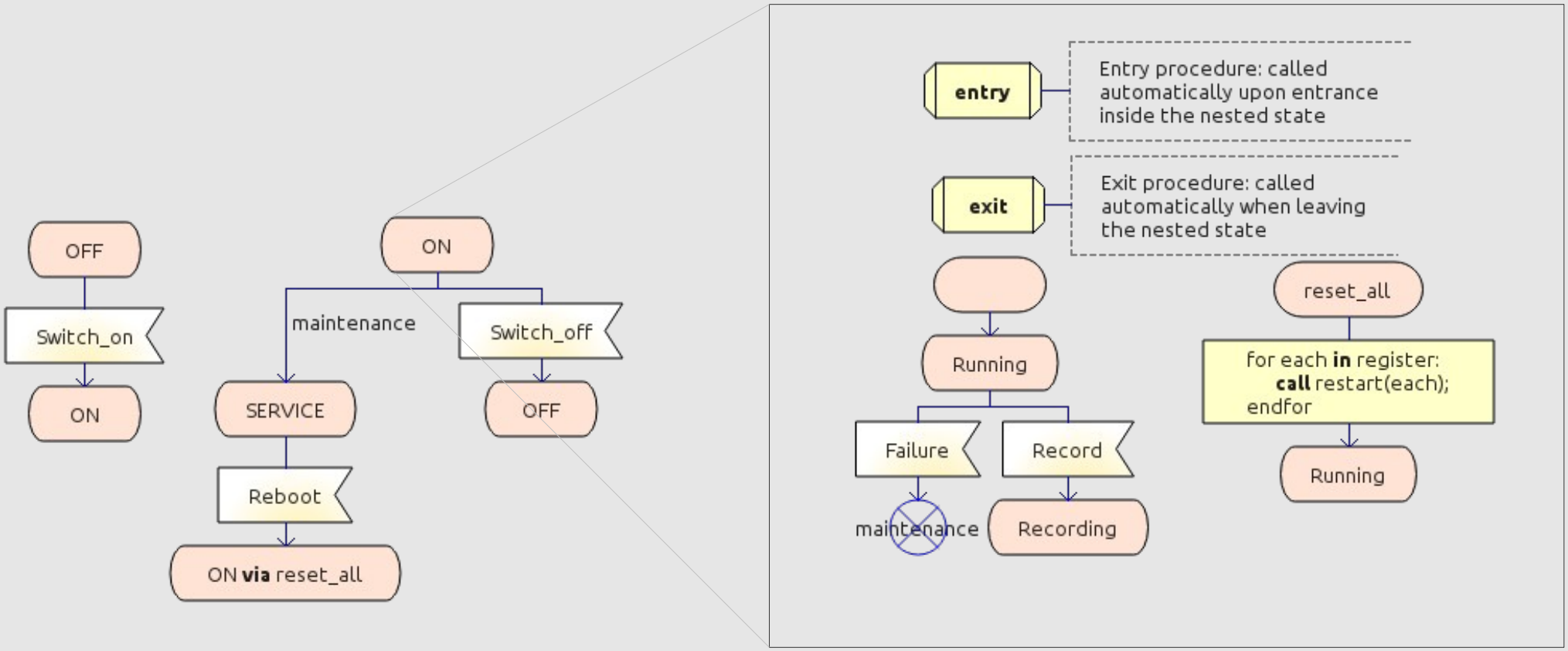
- International, widely used standard (ISO)
- Simple text notation for precise and complete **data type description**
- Real added value : the *physical encoding rules* – read about that !
- ASN.1 types and value notation can be used in SDL – they are part of the language

```
Point ::= SEQUENCE {
    x REAL (-10.0, 10.0),
    y REAL (-10.0, 10.0)
}
myPoint Point ::= { x 5.0, y 1.0 }
Point [encoding IEEE754-1985-64, endianness little]
```

Typical SDL diagram



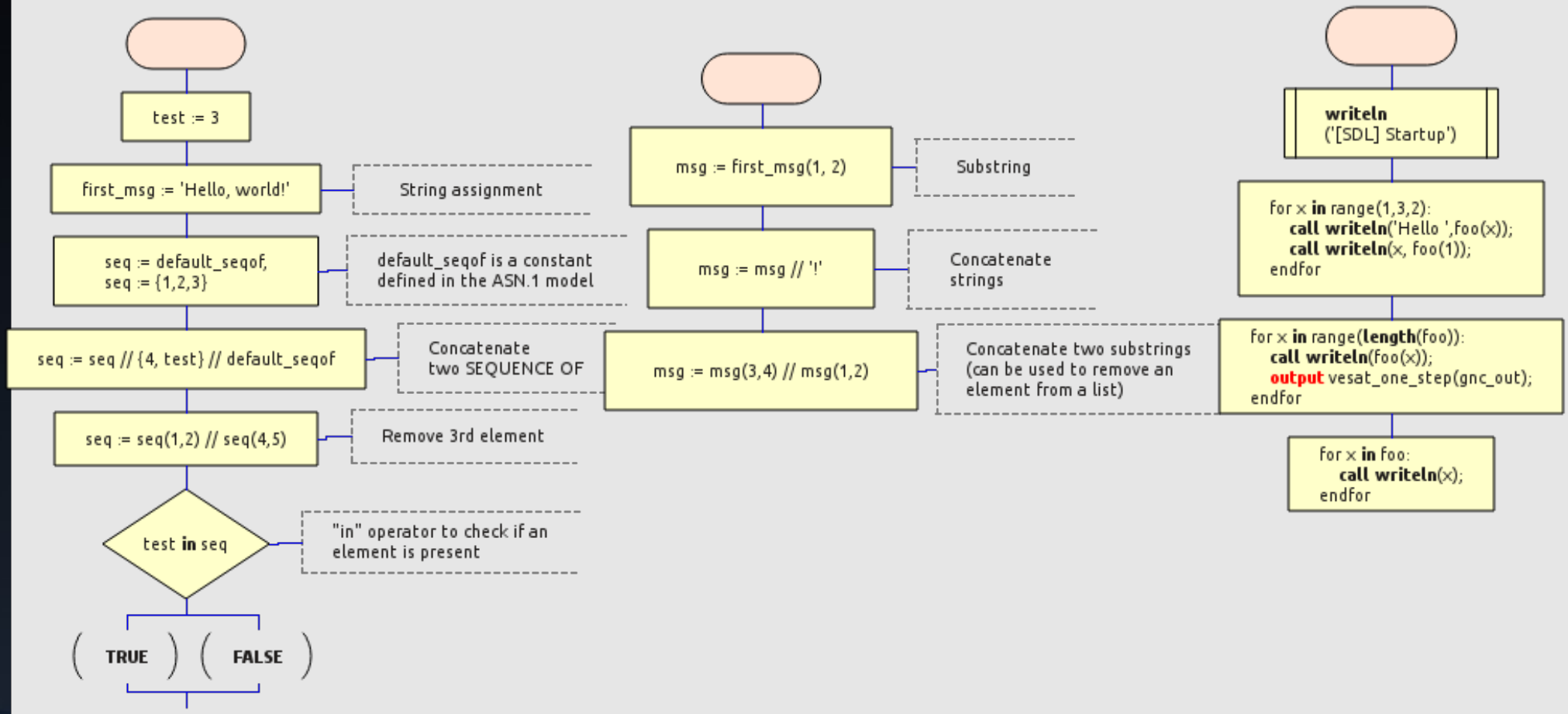
Composite states



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

Advanced data manipulation

- Declare variables of ASN.1 types
- Use strings and arrays with an easy syntax
- Use FOR loops



A great user experience

- Context-dependent autocompletion
- Hyperlinks
- Export diagrams to PNG/PDF files
- Convert SDL to Statechart
- Smooth symbol placement
- Context-dependent activation of symbol icons
- Undo, redo, copy-paste, « vi »-mode, zoom
- Syntax highlighting

But more important : OpenGEODE is designed for safe applications

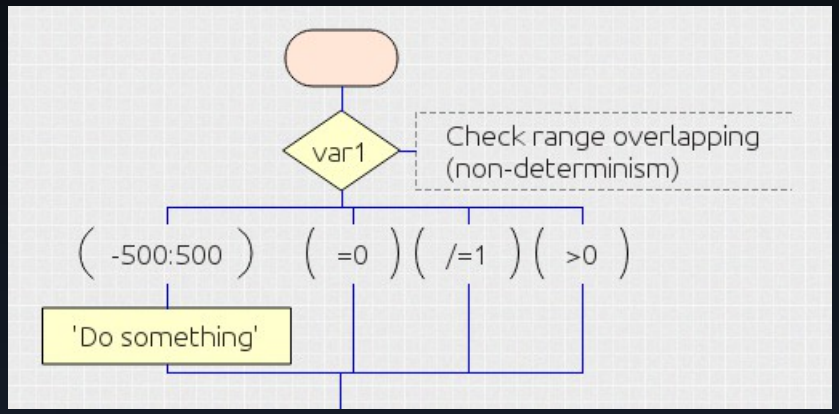
- The syntax and semantic checker is powerful, detecting issues that even Ada or F# compilers do not spot
  - Non-determinism is forbidden
  - Risk of overflows are detected statically
  - Dead code (unreachable branches) are spotted
  - Missing and duplicate answers in a DECISION (switch-case) are detected

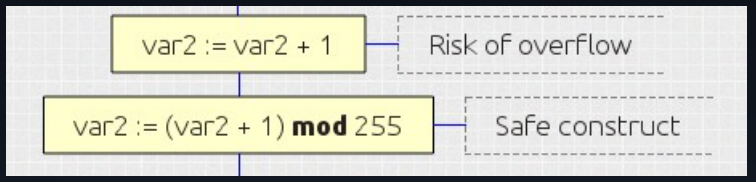
Example of errors the tool catches

Given this variable :

```
dcl var1 INTEGER (-2147483648 .. 2147483647)
dcl var2 INTEGER (0 .. 255)
```

Do not try this :






```
[ERROR] Decision "var1": answers =0 and /=1 are overlapping in range [0 .. 0]
[ERROR] Decision "var1": answers =0 and -500:500 are overlapping in range [0 .. 0]
[ERROR] Decision "var1": answers /=1 and >0 are overlapping in range [2 .. 2147483647]
[ERROR] Decision "var1": answers /=1 and -500:500 are overlapping in range [-500 .. 0]
[ERROR] Decision "var1": answers /=1 and -500:500 are overlapping in range [2 .. 500]
[ERROR] Decision "var1": answers >0 and -500:500 are overlapping in range [1 .. 500]
[ERROR] "var2:= var2 + 1" : Expression in range [1.0..256.0], outside expected range [0..255]
```

Summary on the language

- 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

Backends, benchmarks

- OpenGEODE comes with two backends : Ada and LLVM code generators
  - Target safe code, without heap usage, without external dependencies
  - Integrated with the ASN.1 Compiler from ESA/Neuropublic, generating C and SPARK/Ada code
  - Generated code is simple, readable, and easily customizable, binaries are tiny and speedy

```
Size: Ada 100.00% LLVM 21.33%
Time: Ada 100.00% LLVM 42.29%

| Benchmark        |   Ada size (B) |   LLVM size (B) |   Ada time (us) |   LLVM time (us) |
|------------------+----------------+-----------------+-----------------+------------------|
| test-controlflow |          42808 |            7280 |            2030 |              844 |
| test-exitnested  |          42800 |            6640 |            2018 |              845 |
| test-expressions |          59256 |           19288 |            2036 |              859 |
| test-operators   |          48900 |           10352 |            2087 |              895 |
| test-substrings  |          51104 |           12624 |            2038 |              873 |
| test10           |          44304 |            7768 |            2047 |              873 |
| test11           |          42800 |            7156 |            2009 |              845 |
| test8            |          49664 |           12624 |            2075 |              876 |
```

Maintenance, future work

- OpenGEODE is less than 10,000 lines of code, including everything (parser, graphical editor, backends, checkers)
  - Thanks to Python, Qt, and ANTLR....
  - Code is well documented and easy to maintain
  - Writing new backends is easy, a template is provided
  - Several external contributors « forked » the source code and developed new features
- Ongoing/Future work : verification with model checking, SDL to VHDL backend, simulation, more checks, improved SDL to statechart conversion, ... Use on a space project ?