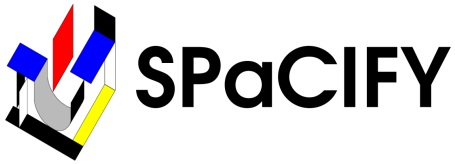


- SPaCIFY project - Synoptic : Spacecraft Synchronous DSML

Alexandre Cortier
Post-doc at IRIT/ACADIE
cortier@irit.fr

SPaCIFY : ANR Project

Spacify ANR (French Research National Agency) Project
End : September 2009

- Aim :
  - R&D project aiming at developing a design environment for critical embedded software (Spacecraft System)
  - Model-Driven Engineering and Formal Methods
    - model checking
    - formally-verified domain specific transformations
  - multi-clock synchronous paradigm
  - simulation and analysis tools
  - scheduling analysis
  - executive platform supporting distribution, partitionning and dynamic adaptation (middleware)

Environment components will be built upon the Topcased toolkit. (*The Open-Source Toolkit for Critical Systems*)

The SPaCIFY Project

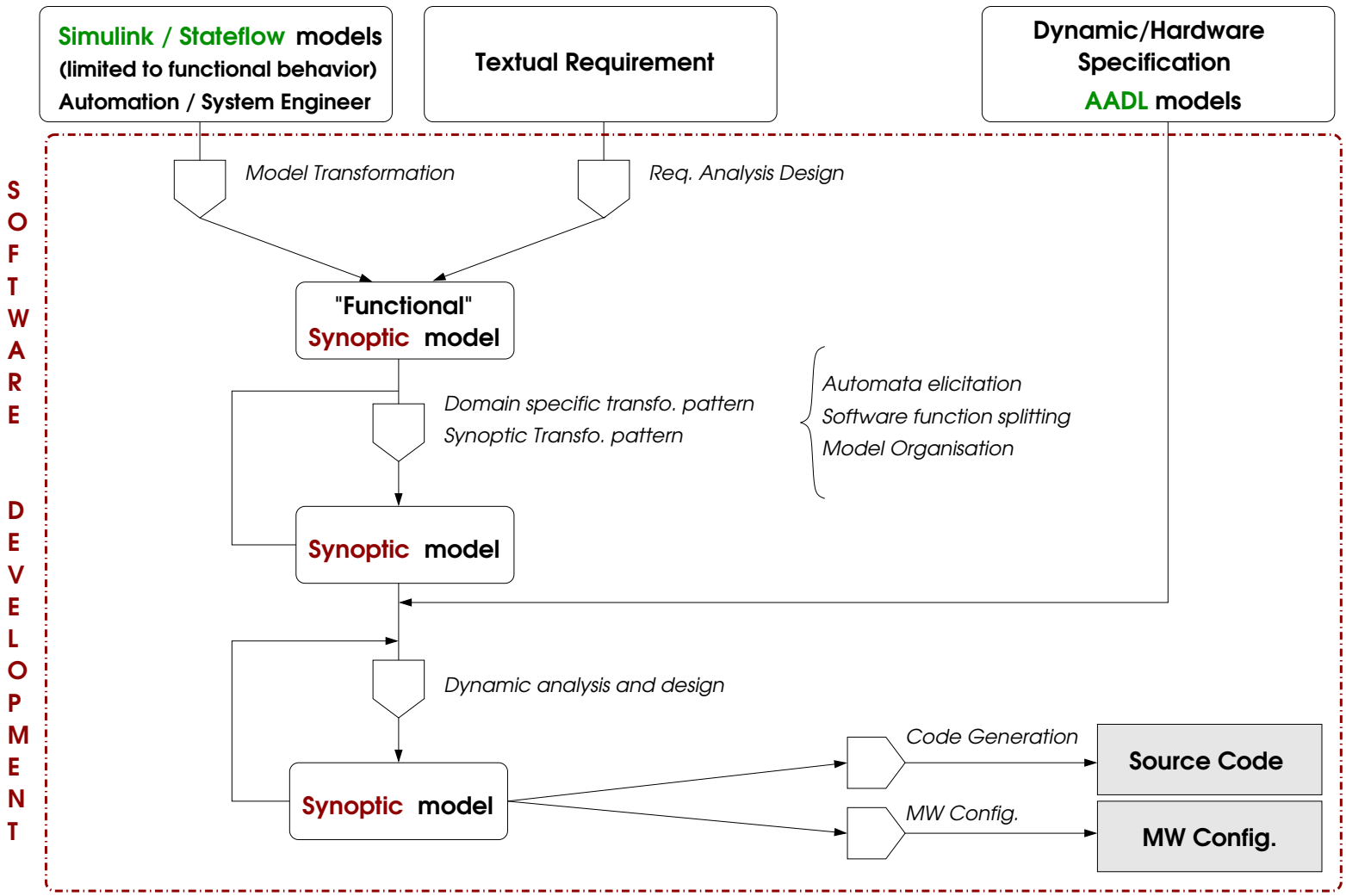
- Industrial Stakeholders :
  - CNES, Thales Alenia Space, EADS Astrium
    - spacecraft system designers
  - Anyware Technologies, GEENSYS
    - graphical design environment
    - configuration, versions and documentation management
- Academic Stakeholders :
  - IRIT-ACADIE (Toulouse): Synoptic language definition, formally-verified model transformations
  - ENSTB-CAMA (Brest) : middleware design
  - IRISA-ESPRESSO (Rennes) : synchronous semantics
  - LaBRI (Bordeaux) : model-checking

Synoptic

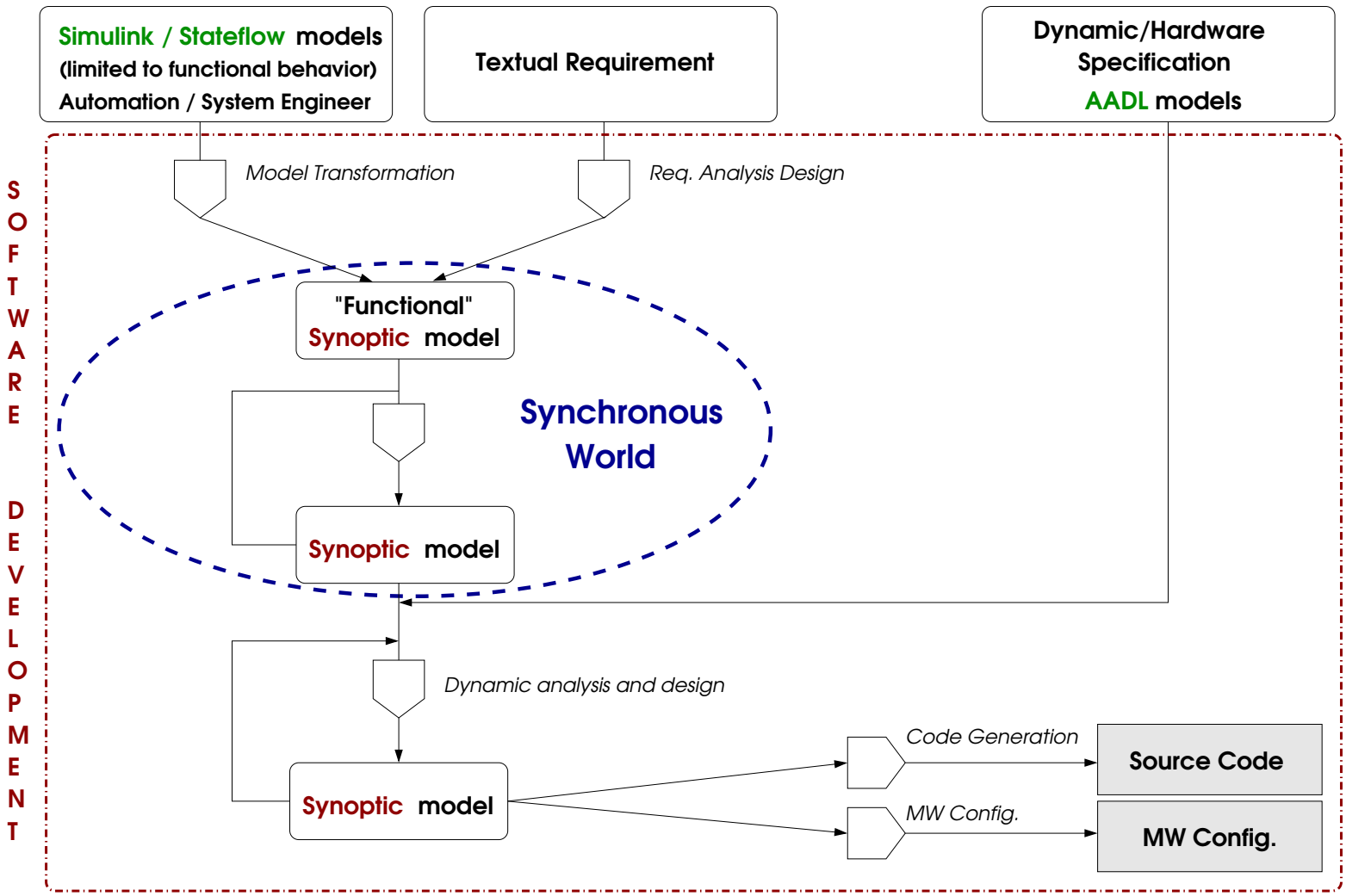
Synoptic : DSML (*Domain Specific Modeling Language*) for spacecraft systems.
Requirements :

- central language of the development environment
- support an iterative and incremental development process
- functional, architectural and dynamic views specifications
- modular design
- based on a synchronous semantics (functional part)

Overview : SPaCIFY development process



Overview : SPaCIFY development process



Synoptic

Synoptic :

- a graphical and textual DSML
- provides high-level constructions to handle
  - multi-layers description (various modelling aspect)
  - various granularity levels (iterative and refinement development)
  - modular approach
- based on a synchronous semantics.
  - formal and deterministic analysis and verification
  - refinement proof
  - transformation proof

Synoptic : multi-layers system specification

Synoptic is not fundamentally a new language but an integration of different sources and concepts.

Synoptic is inspired by several approaches :

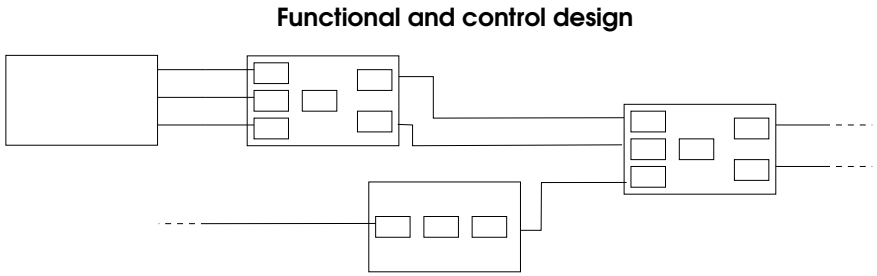
- **Geneauto** : safe subset of the Simulink/Stateflow modelling language used for the development of certified safety critical embedded real time systems
- **AADL** : *Architecture Analysis & Design Language* (formerly Avionics Architecture Description Language)
- **Components Models** : CCM, Fractal

Synoptic : multi-layers system specification

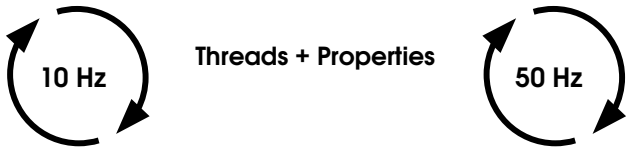
- Software Architecture : **Geneauto approach**
  - structural feature : Dataflow models (“Blocks Diagrams”)
  - behavioral feature : Control Flow models (“Finite States Machines”)
  - real-time constraints : clock properties
- Dynamic/Hardware Architecture : **AADL approach**
  - Threads description
  - platform aspects (“components view”)
  - mappings : which component execute which functional blocks ?
    - functional blocks $\longrightarrow$ threads
    - threads $\longrightarrow$ hardware components (processor)
    - signals $\longrightarrow$ bus
    - variables $\longrightarrow$ memory

Synoptic : multi-layers system specification

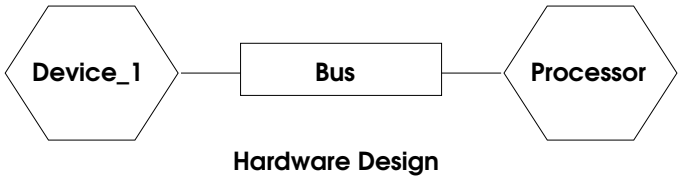
Software architecture



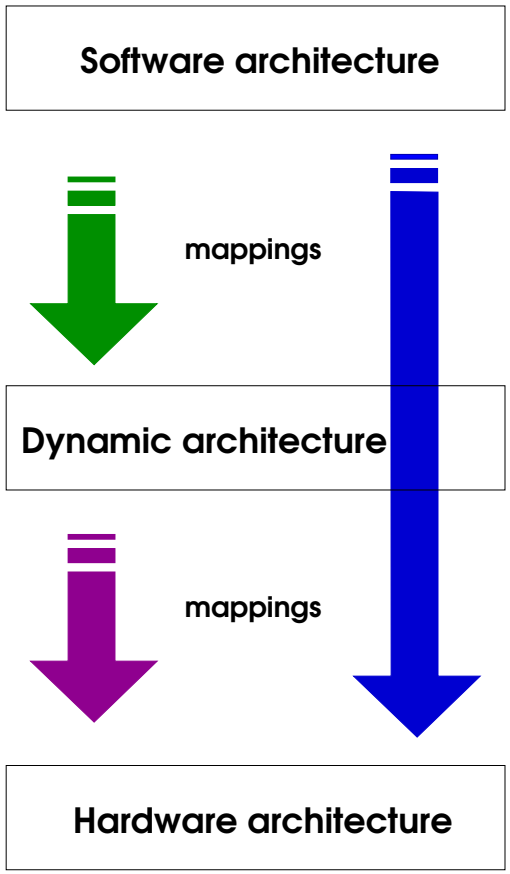
Dynamic architecture

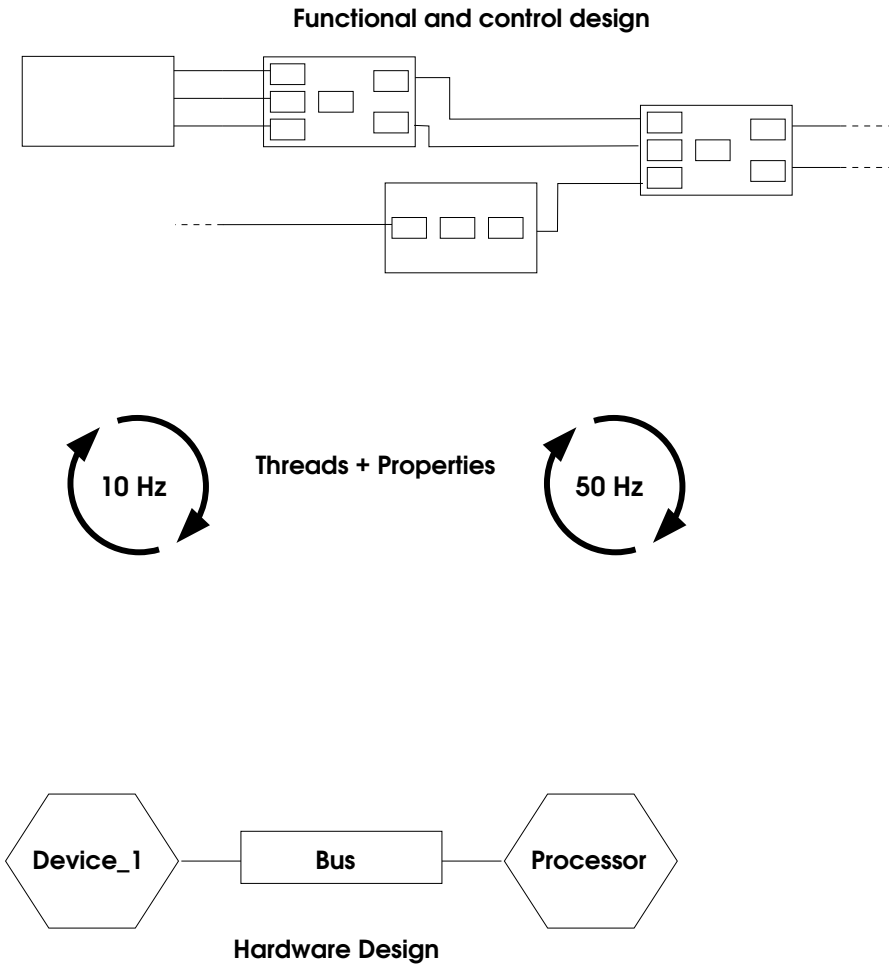


Hardware architecture

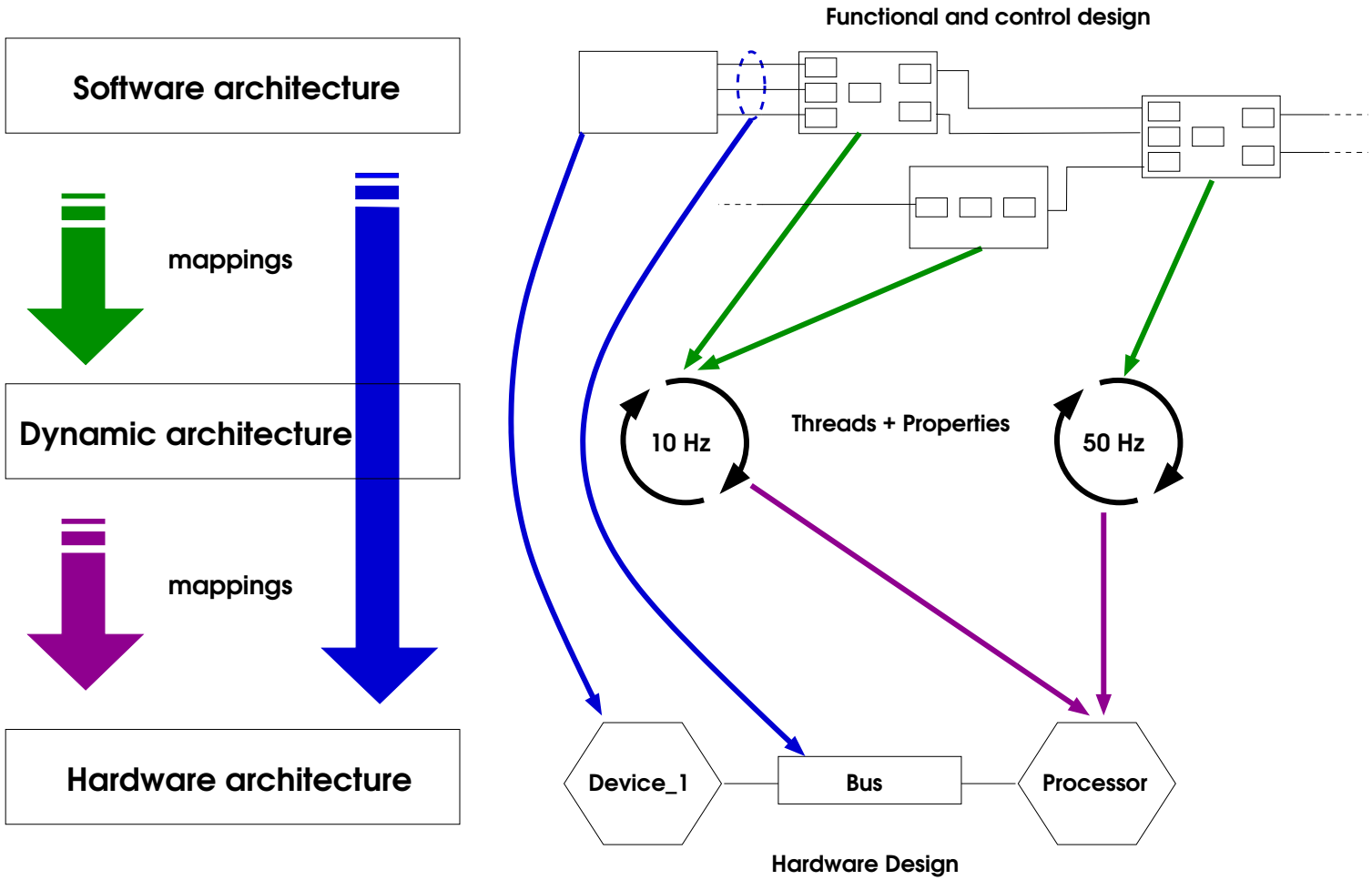


Synoptic : multi-layers system specification





Synoptic : multi-layers system specification

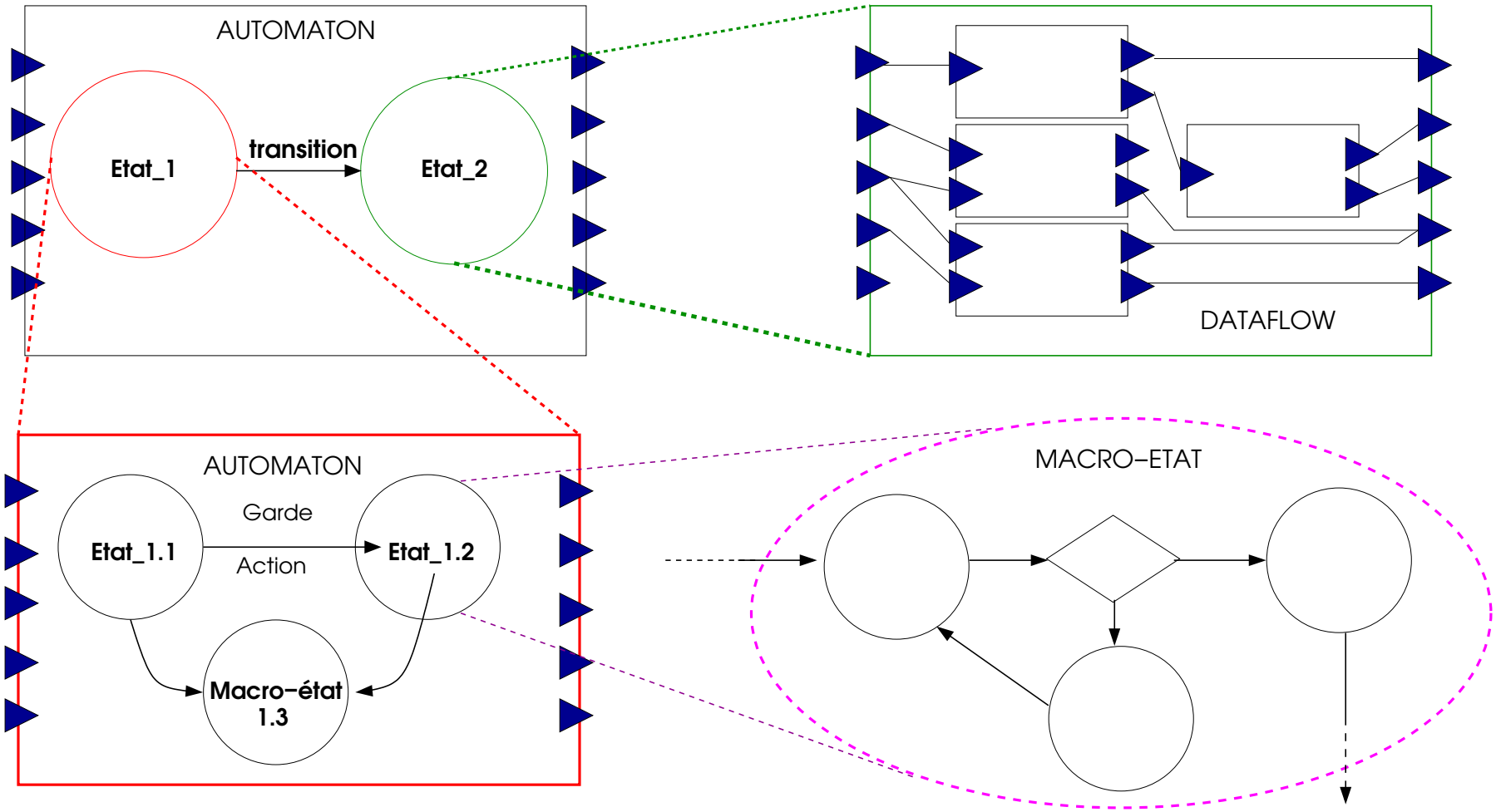


Synoptic : functional model

- Software architecture = blocks/nodes hierarchy
- A node in a block diagram has :
  - a **type**
  - several **implementations**
- a **node type** describes interaction ports (interface)
- different kinds of implementations :
  - **dataflow** : describes functional part
  - **automaton** : describes behavioral part (modes)
  - **external/primitive** : “black box”

Dataflow and automaton blocks are mutually nested.

Synoptic : functional model (block hierarchy)



Synoptic : functional model (**node type**)

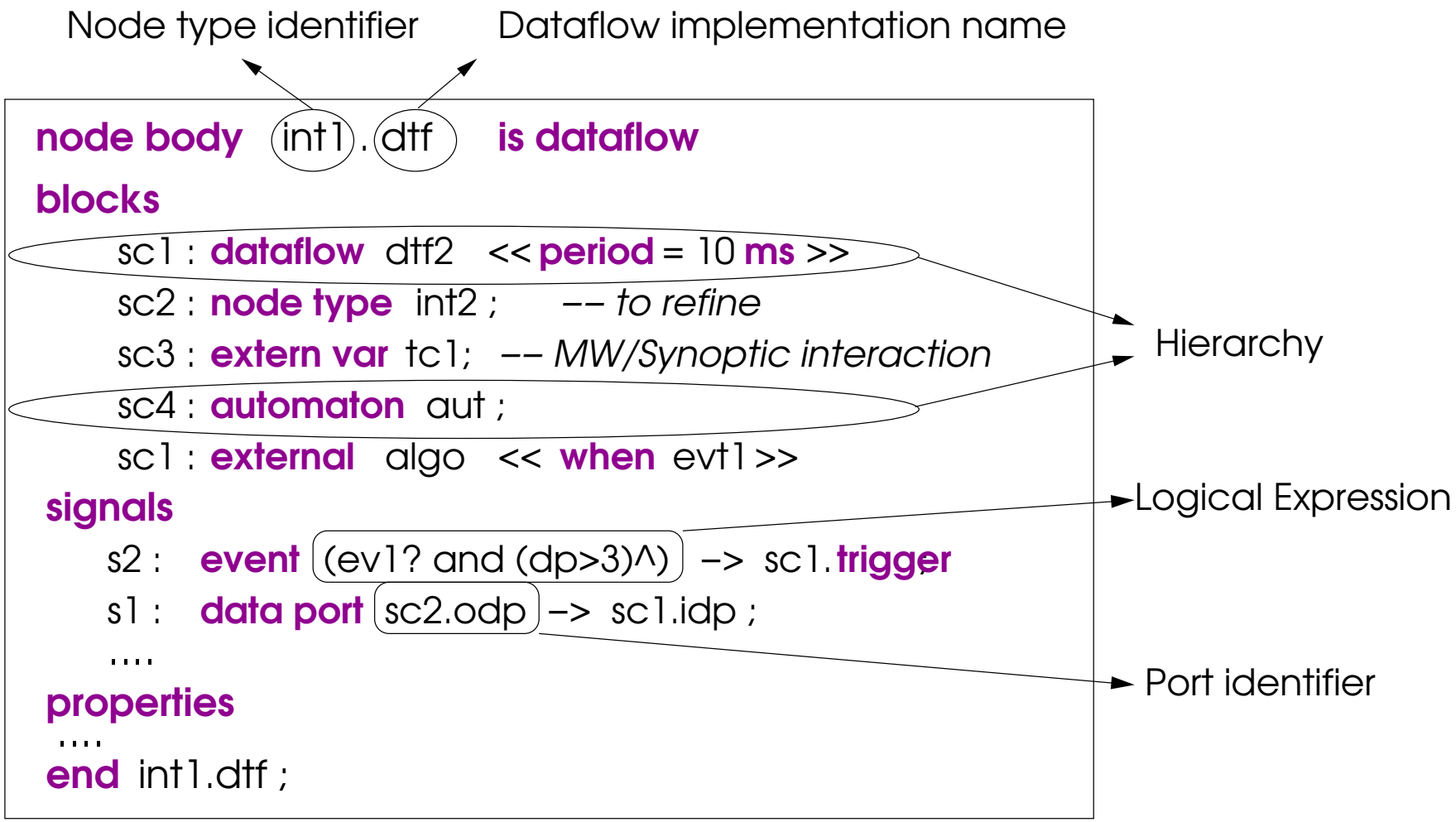
Node type example :

```
1 node type SunPointing
2 features
3 IMU_Data: in data port  array 3 of double;
4 STR_Data: in data port  array 4 of double;
5     DOR_Data: in data port array 4 of double;
6     MTQ_Cmd: out data port  array 3 of double;
7     RW_Cmd: out data port  array 3 of double;
8  end SunPointing;
```

Implicit ports :

- **reset** : re-initialization (boolean port)
- **trigger** : block activation (event port)
- **enable** : block activation control (boolean port)

Synoptic : functional model (**dataflow**)



Synoptic : functional model (**automaton**)

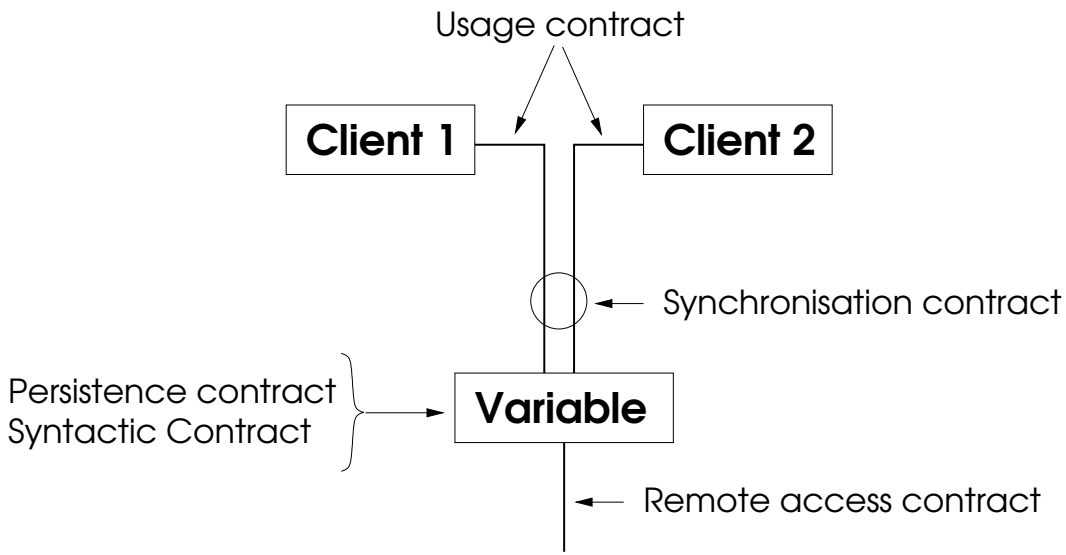
```
automaton INIT_calculateur.aut
 states
    PM_RESET: state dtf, /* dtf = datfalow implementation */
    Boostrap: state,
    OBSW_running : state,
    Firmware: macro state
              states
                PM_Built_IN_TESTS: state,
                OBSW_checkers: state;
              initial state PM_Built_IN_TESTS;
              transitions
                tr1:PM_Built_IN_TESTS-[ ]→OBSW_checkers;
        end Firmware,
    RAM: macro state [...]  end RAM;
 initial state PM_RESET;
 transitions
     t1 : PM_RESET -[on G do A ]→ Firmware.PM_Built_IN_TESTS;
     t2 : Firmware.PM_Built_IN_TESTS-[ ]→Firmware.OBSW_checkers;
     [...]
end INIT_calculateur.aut;
```

Synoptic/MW : external variables

The Middleware has to abstract the asynchronous behavior of the system (bufferisation,...)
Interactions between MW and Synoptic models are handled using **external variables** concept.

- external variables = sources / sinks of signals
- external variables types :
  - constants, TM, TC, global variables,
- external variables contracts

Synoptic/MW : external variables

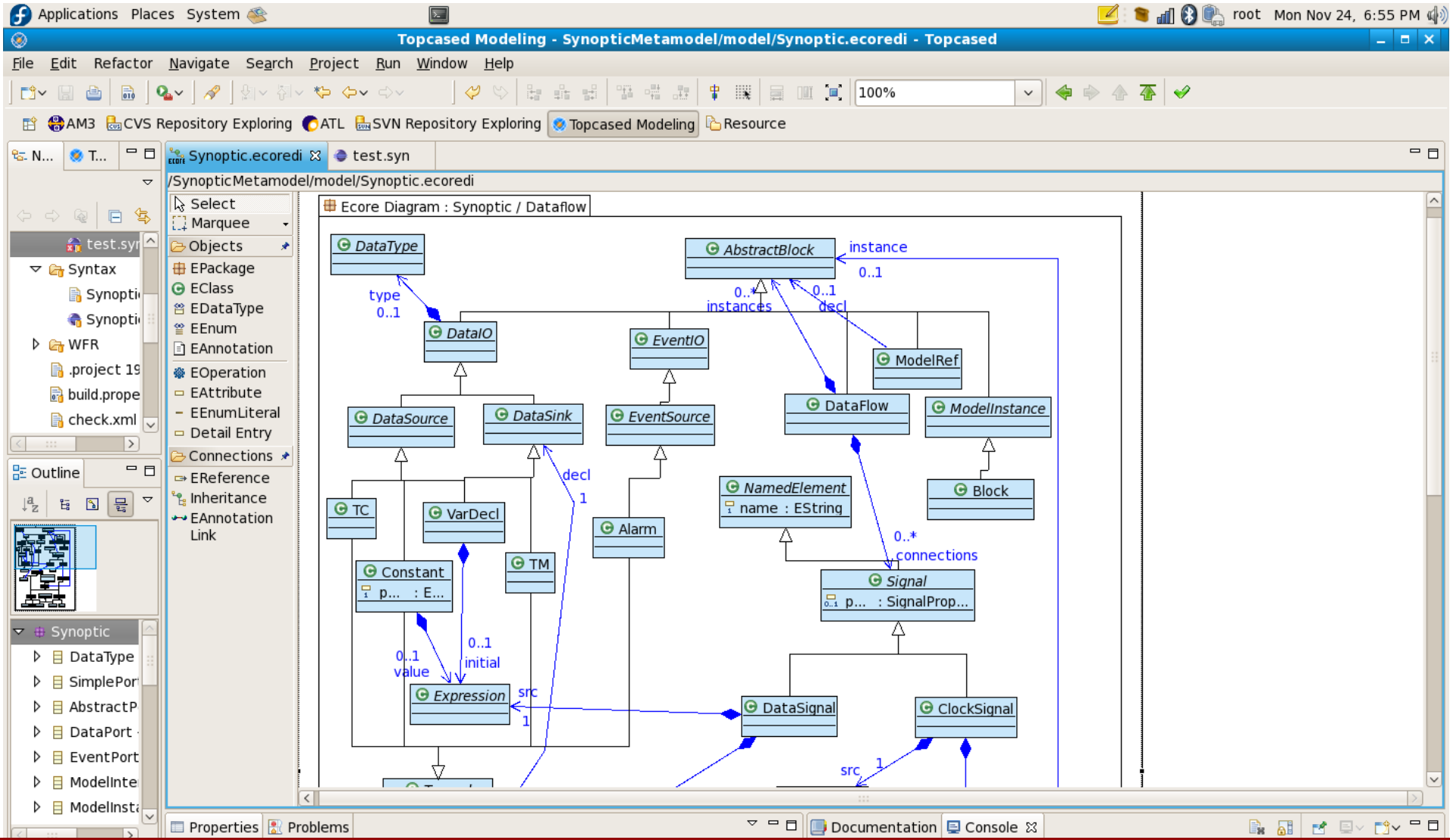


External variables and associated contracts are used to configure the MW

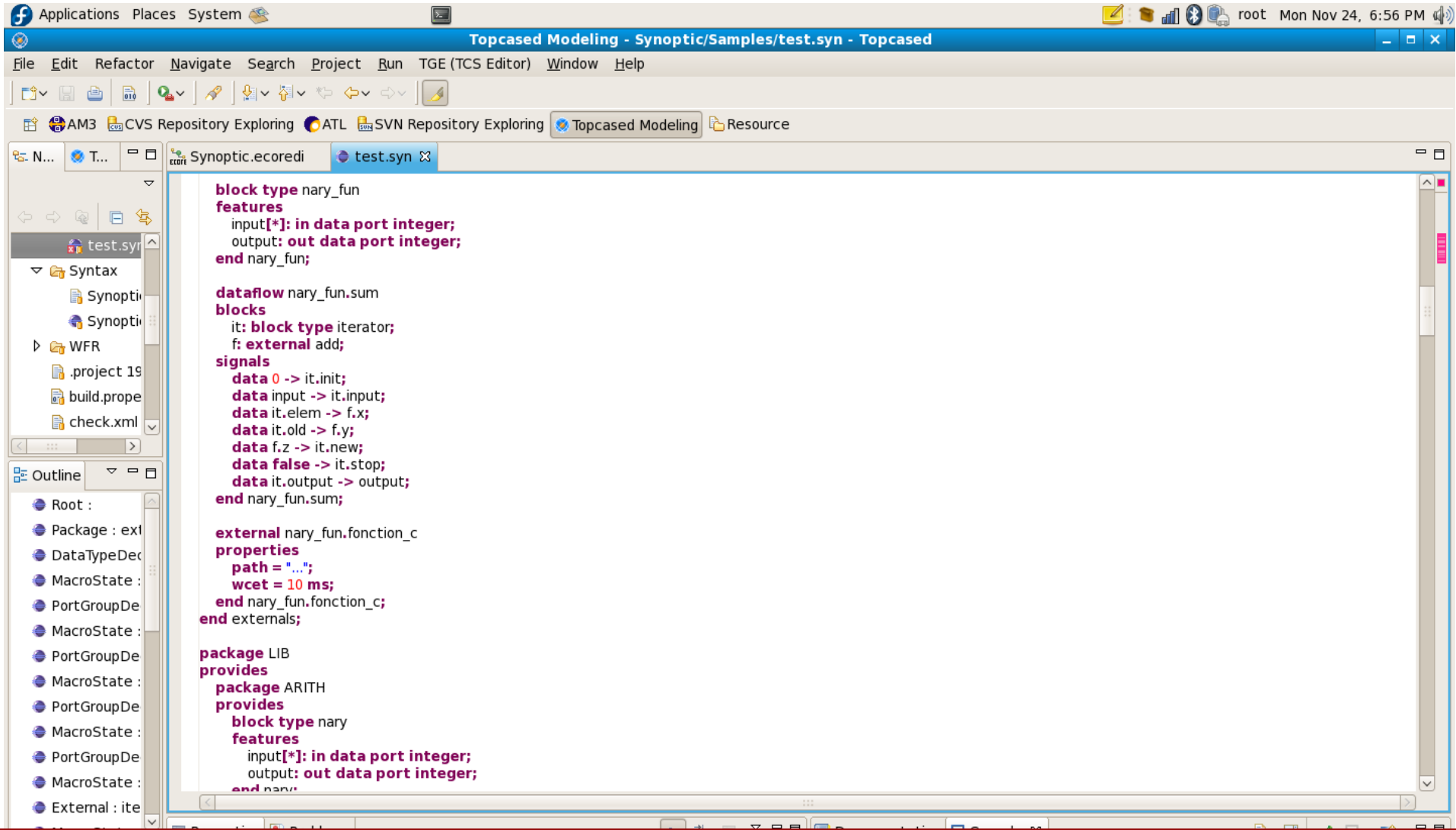
Synoptic Language Definition

- Synoptic Language Definition
  - Meta-Model definition (last steps)
  - concrete textual syntax
- Synoptic Semantics ?
  - in progress...
  - semantics by traduction (IRISA-ESPRESSO) : Signal (Polychrony)
  - we need to define a semantics for the language core
    - works on transformations proof (Martin Streker)

Meta-Model Ecore



Textual Syntax and Edition tool (Topcased)



Transformations Validation

- Automatic mappings : blocks → threads
  - to assist the developper
  - need a formalisation of current pragmatic rules used by system engineers
- Blocks and Signals Refinements :
  - automatic refinement (patterns : validated transformation)
  - Proof Obligation (PO) generation for manual refinements
- Edition transformations :
  - ex : Model organisation, Software function splitting
  - Automatic and validated transformations

Questions ?

Questions ?