Model-Checking Support in TOPCASED

S. Dal Zilio, F. Vernadat, B. Berthomieu (LAAS)

M. Filali, J.-P. Bodeveix (IRIT)

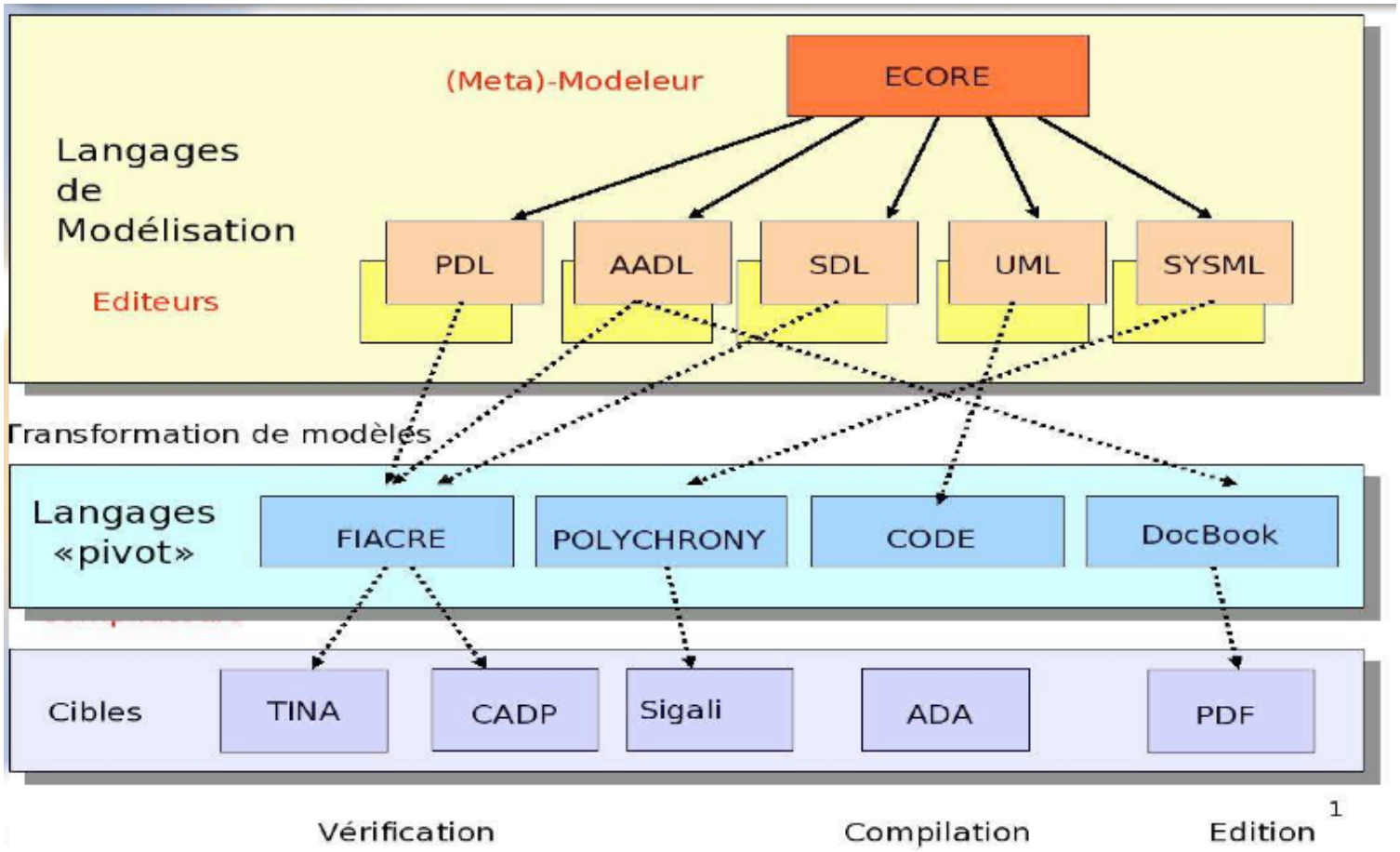
work partially supported by the French FNRAE project QUARTEFT (http://quarteft.loria.fr)

A talk in two parts

1. Extending a formal specification language with real-time properties above/using tracing capabilities

2. A model-checking (transformational) tool-chain for AADL with the BA-annex

Integration within Topcased



TOPCASED
The Open Source toolkit for critical systems

FIACRE

Fiacre 101

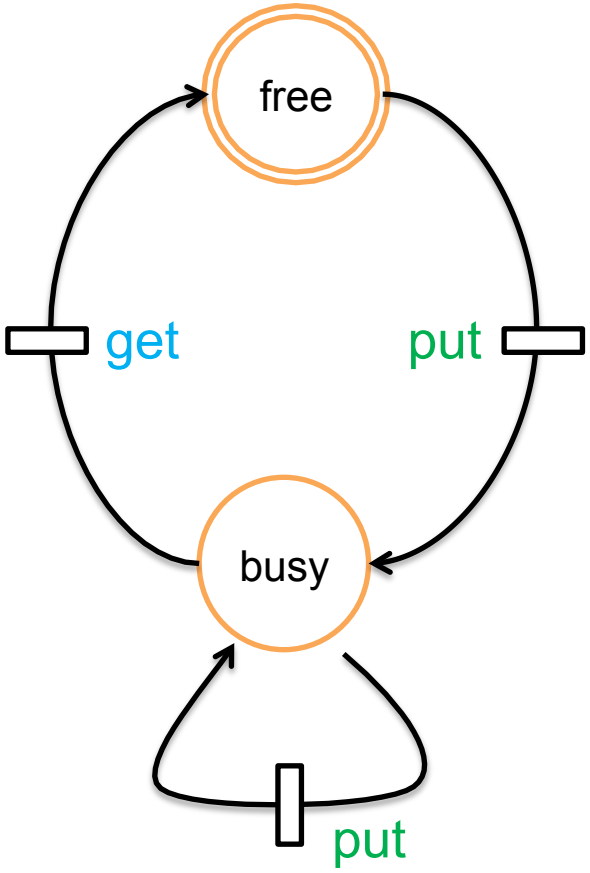
- A formal specification language:
  - concurrency; resources; priorities; timing constraints
- An intermediate language from “business languages” to the entry language of model-checking tools

Fiacre 101

- Processes
  - state machines with expressions denoting symbolic transitions
  - Both shared memory and message-passing
  - data are statically typed
- Components
  - unit for defining interactions, priorities and timing constraints

A simple example (process)

```
process lbuf [put : none, get : none] is
        states busy, free
        init free
        from free   put; to busy
        from busy
             . select
                    put; to busy
               []   get; to free
               end
```



A simple example (component)

```
// component

component Main [ ports ] ( &var s)   is

   priority p > q , …

   port req: request, resp: nat , pv : … in ]2.4 , 6]

   par  Worker[resp,req]

     ||  (par  pv → lbuf[req,pv] || pv → lbuf[pv,resp] end)

   end
```

Recent Extensions

- additions to the language
  - non-“time resetting” transitions ; extended priorities definitions ; …
- additions to the toolbox
  - symbolic methods ; partial exploration ; concurrent implementation ; database ; …
- Libraries of “real-time architectural patterns”, e.g. periodic processes

Recent Extensions

- Properties and "observables"
  - probes = declares what is observable in a system (state, data-expression, events, tag, enter, leave)
  - *observers* = a distinguished class of components
  - An assertion language based on patterns

Property patterns

- General properties
  - deadlockfree
  - infinitelyoften o
  - mortal o

Property patterns (present)

present o

present o after o’

present o before o’

present o between o_1 and o_2

present o after o_1 until o_2

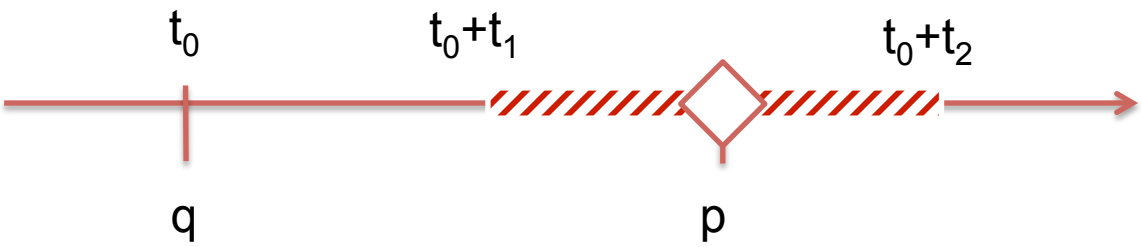
see e.g. http://patterns.projects.cis.ksu.edu/

Property patterns (other classes)

- Absence: absent o
- Response: o_1 leadsto o_2
- Universality: always o
- Precedence: o_1 precedes o_2
- Composition: P and Q ; not P ; P $\multimap$ Q

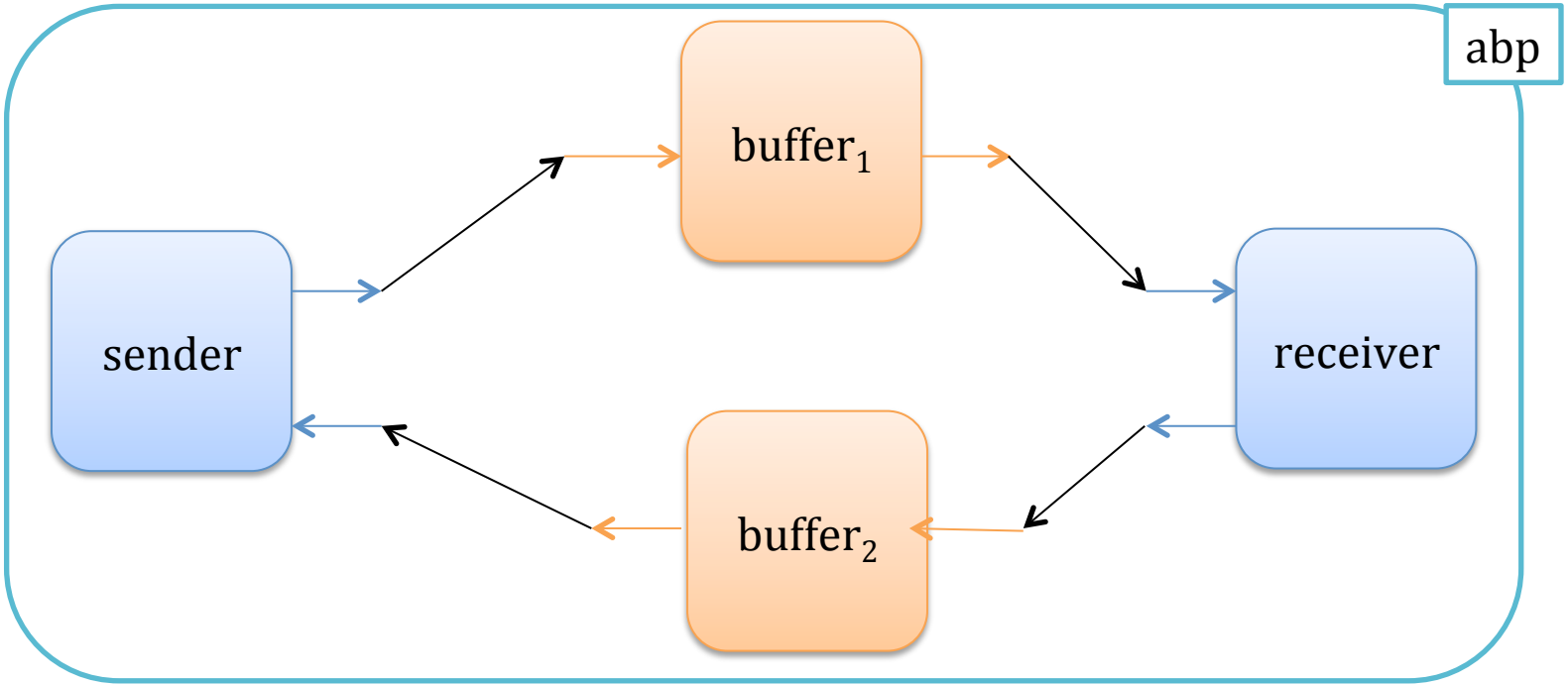
Property patterns (real-time)

present p after q within $[t_1;t_2)$



present p after q lasting d

Probes



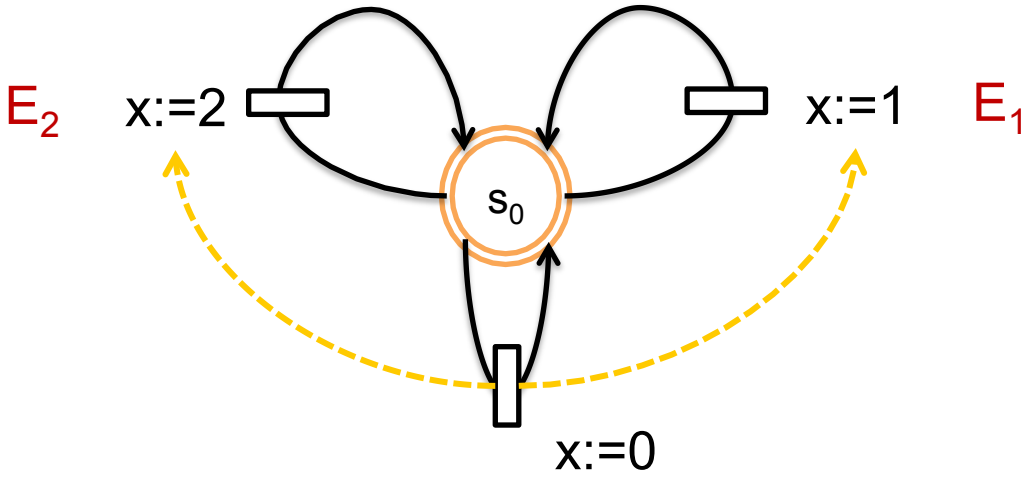

```
property safe is ltl []
   ((abp/1/state send => abp/2/value (empty buff)) and
    (abp/4/state ack => abp/3/value (empty buff)))

assert safe
```

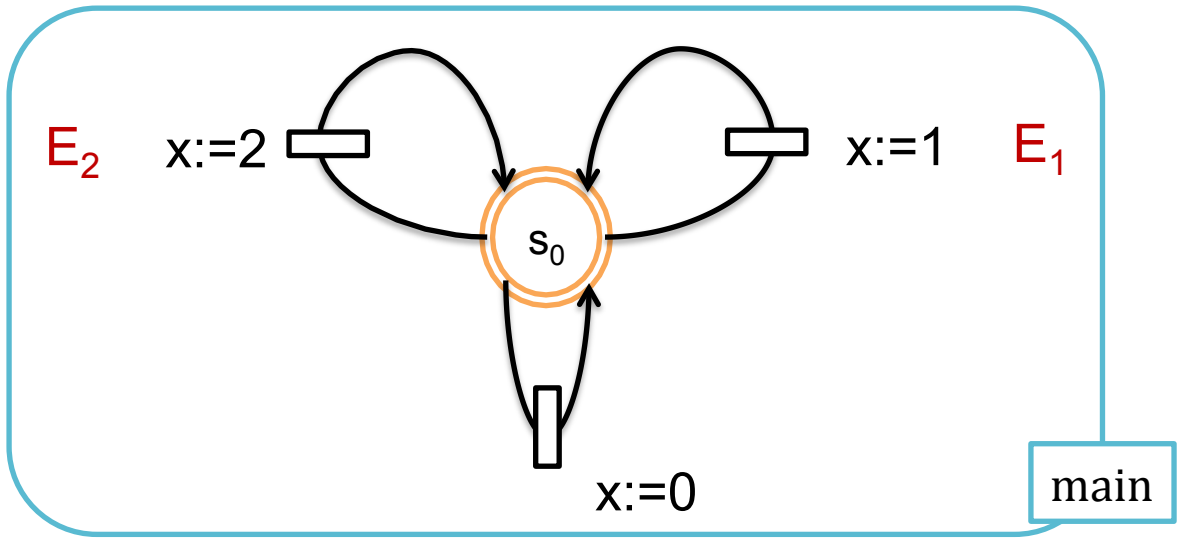
```
process universal(&x : nat) is
   states s0
   from s0
      select
         x := 1; to s0
      [] x := 2; to s0
      unless
         on (x <> 0) ; wait [0,0]; x := 0 ; to s0
      end

component main is
   var x : nat := 0
   par universal(&x) end

main
```



present E_1 after E_2 within [4, 5)




```
component obs is
   port E1 : sync in [0,0] is main/1/value (x = 1),
      E2 : sync in [0,0] is main/1/value (x = 2)
   par observer[E1, E2] end

obs > main
```

Observer

```
process observer [E1, E2 : sync] is
   states idle, start, watch, error, stop
   from idle
      E2;
      to start
   from start
      wait [4,4];
      to watch
   from watch
      select
         E1; to stop
      unless
         wait [1,...[;
         to error
      end
```

present E_1 after E_2 within [4, 5)

Observer

```
component obs is
   port E1 : sync in [0,0] is main/1/value (x = 1),
      E2 : sync in [0,0] is main/1/value (x = 2)
   par observer[E1, E2] end

obs > main

property safe is absent obs/1/state error

assert safe
```

AADL2FIACRE

AADL Execution Model

- **Main features**
  - Precise semantics → schedulability, resource usage analysis
  - Well defined thread life cycle : state diagram
  - Communication : read/write timings
  - Mode change timings

AADL Execution Model

- **Synchronous subset**
  - Periodic threads
  - Deterministic data port communications (immediate, delayed)
  - Planned mode change protocol

→ this subset is deterministic (scheduler, execution platform independent)

Interpretation of AADL in Fiacre

- Software elements and devices
- Data types
- Periodic, sporadic, aperiodic threads
- Shared variables, data, event (data) communications
- Port connection protocols
- Mono processor fixed priority scheduler
- Port urgency and queue size
- Behavioral annex attached to threads and devices

Interpretation of AADL in Fiacre

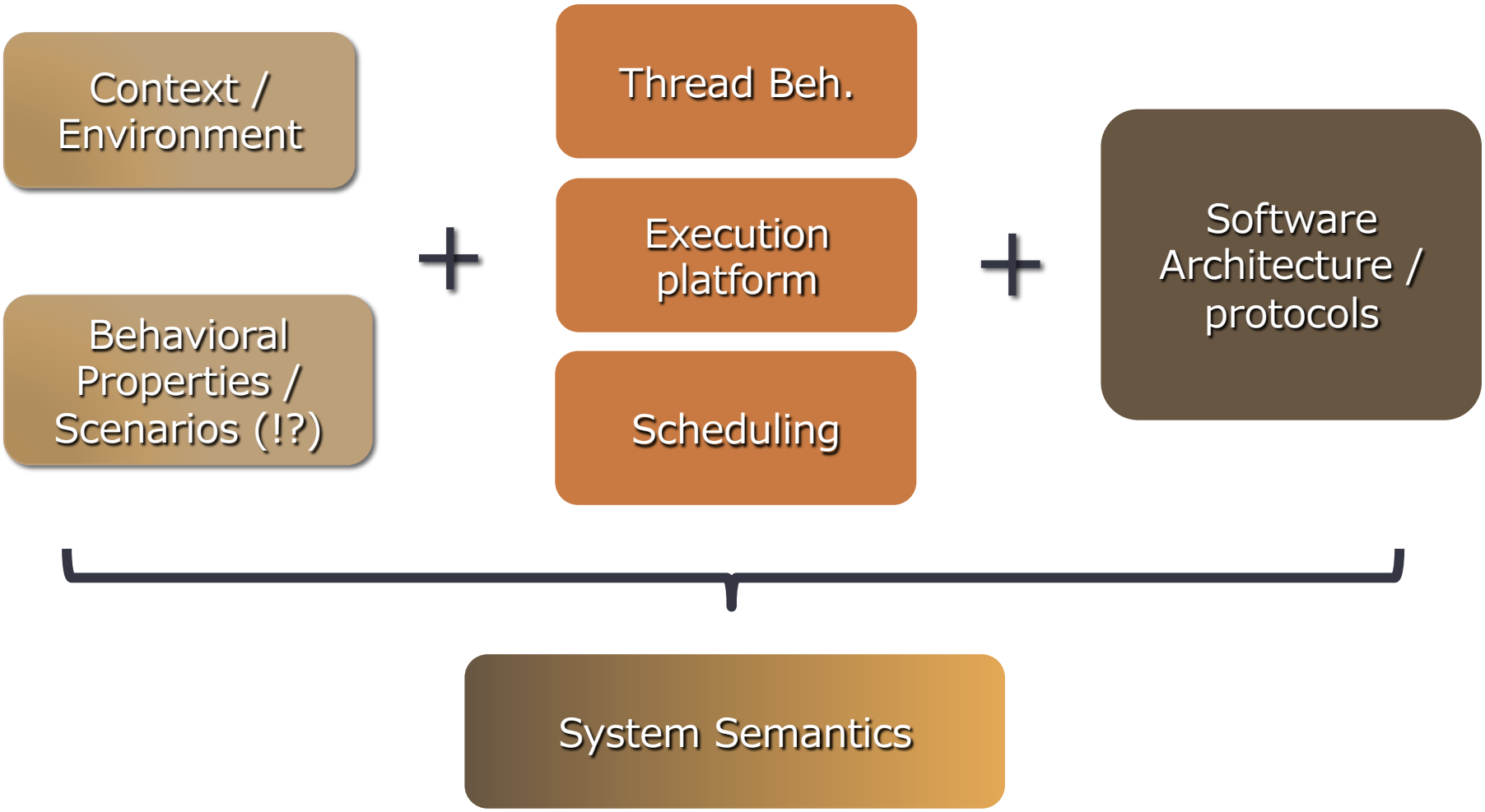
- Verified properties (internalized)
  - Schedulability (data dependent)
  - Buffer overflows (event, event data ports)
- User defined properties (using observers or temporal logic)

Translation Principles

- A controller is attached to each user thread
  - manages dispatch ; check deadline
- A process is attached to each port
  - manage buffers ; synchronizes data transfers with dispatch, completion or deadline
- Each thread executes a loop :
  - wait for dispatch ; gets the processor ; perform user defined computations (behavior annex) ; release the processor (completion)

Use of generic protocol libraries

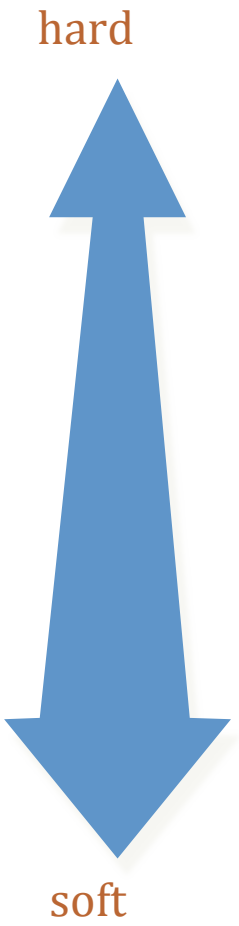
```
process periodic_controller[d: none, c: none,
     dl: in none, w: none] is  /* w: periodic port */
states s0, schedule_error
var st: p_state := p_rdy
init to s0
from s0
  select
     c; st := p_idle; loop
  [] on st=p_rdy; dl; loop
  [] d; loop
  [] w; st := st=p_rdy?p_err:p_rdy; to s0
  unless
     on st=p_err; wait [0,0]; to schedule_error
  end
```

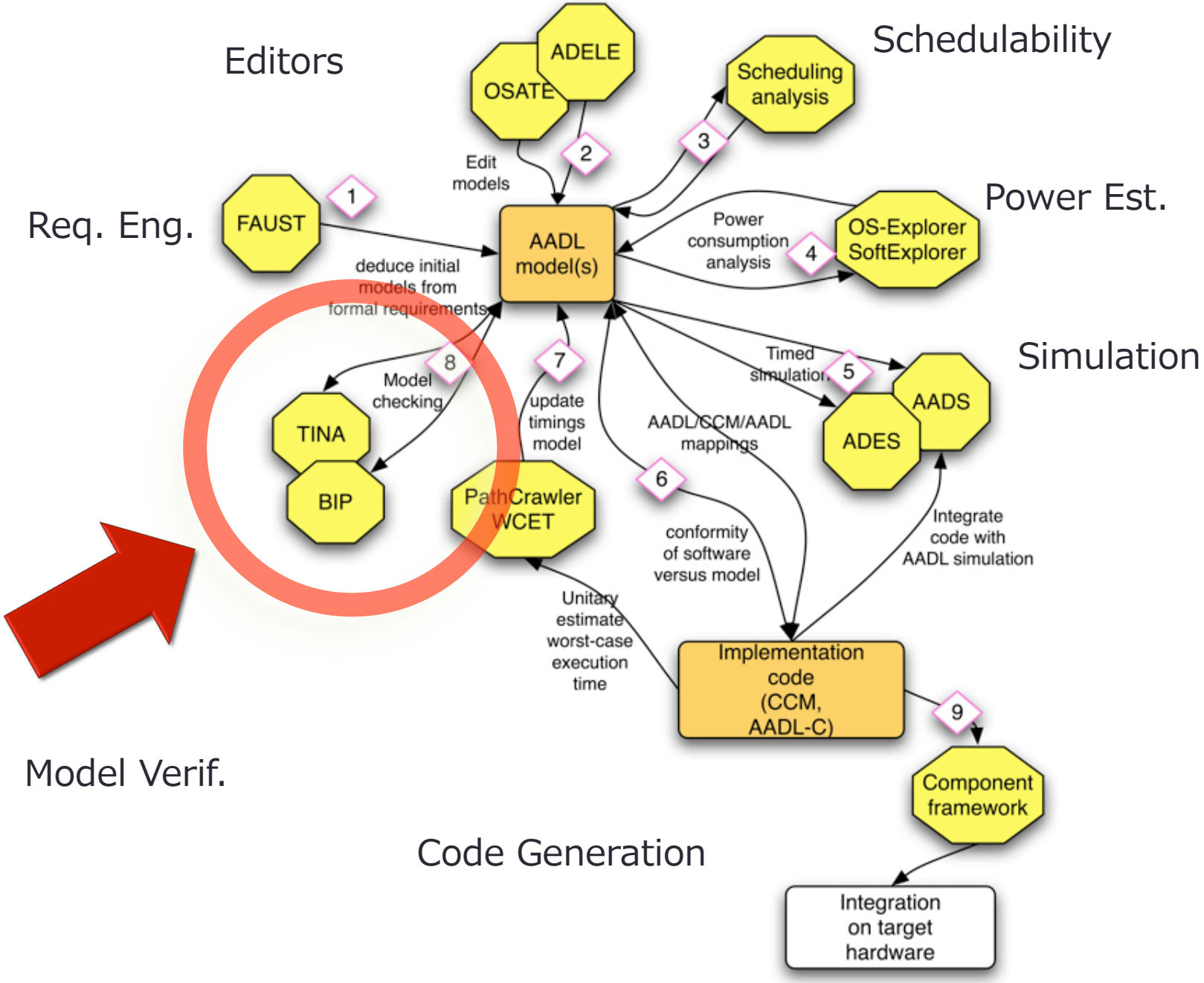

Context / Environment
Behavioral Properties / Scenarios (!?)
+
Thread Beh.
Execution platform
Scheduling
+
Software Architecture / protocols
System Semantics

USE CASES

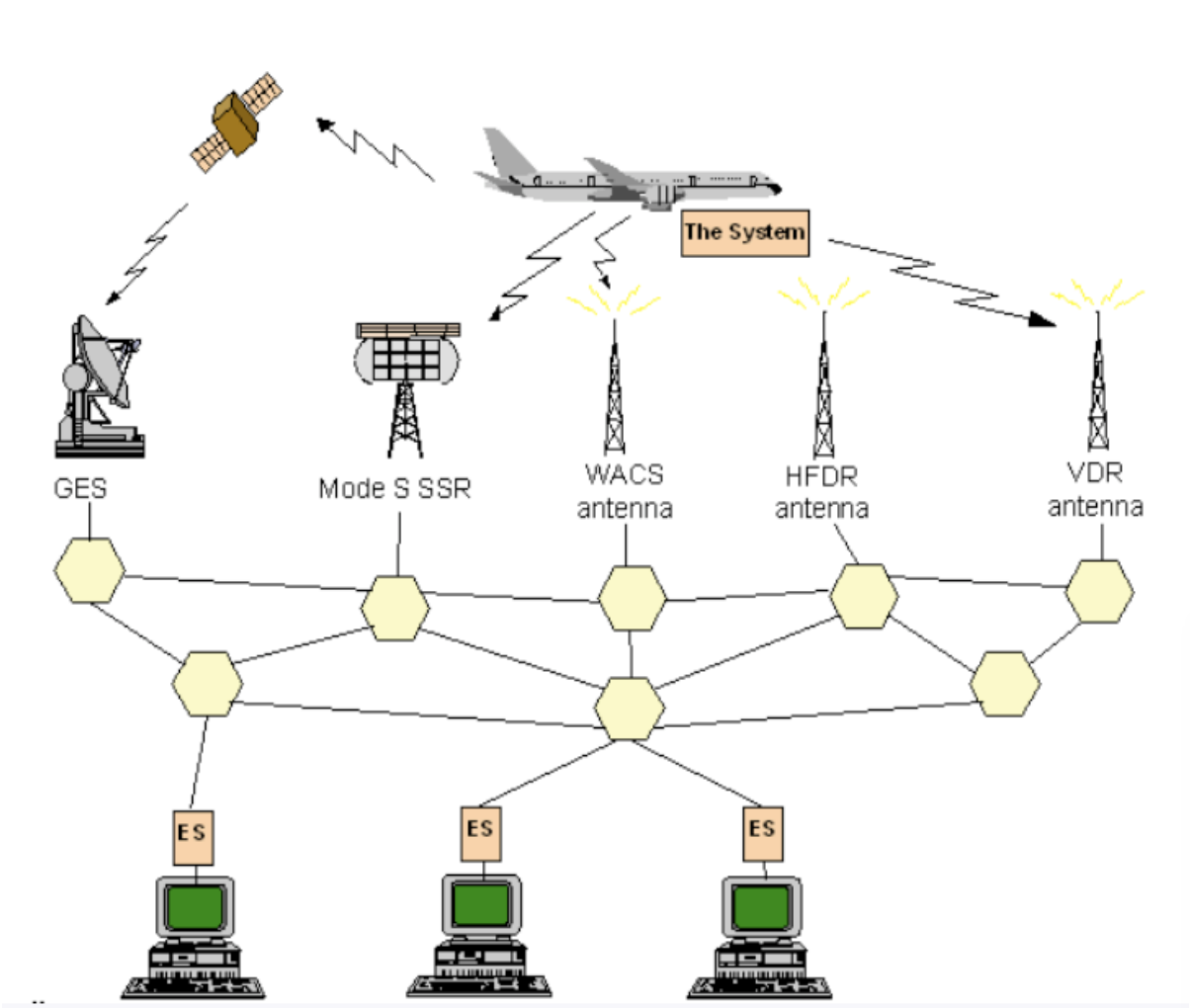
Use Cases

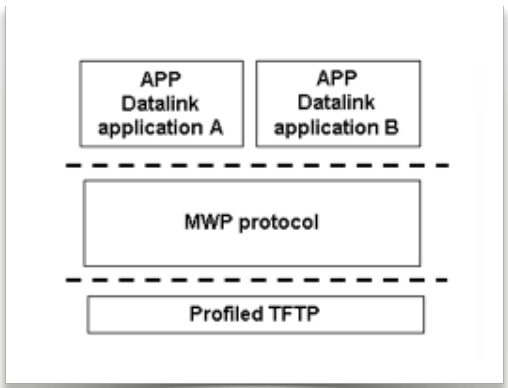
- Flight Warning Systems
- Air/Ground Network Protocol
- Software defined radios

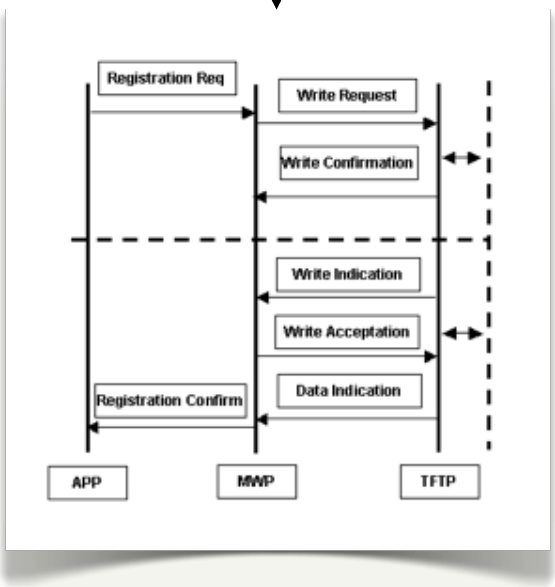



Editors
Schedulability
Req. Eng.
Power Est.
Simulation
Model Verif.
Code Generation
ADELE
OSATE
Scheduling analysis
Edit models
2
3
1
FAUST
AADL model(s)
Power consumption analysis
4
OS-Explorer SoftExplorer
deduce initial models from formal requirements
8
7
Model checking
update timings model
Timed simulation
5
AADS
ADES
TINA
BIP
PathCrawler WCET
AADL/CCM/AADL mappings
6
conformity of software versus model
Integrate code with AADL simulation
Unitary estimate worst-case execution time
Implementation code (CCM, AADL-C)
9
Component framework
Integration on target hardware

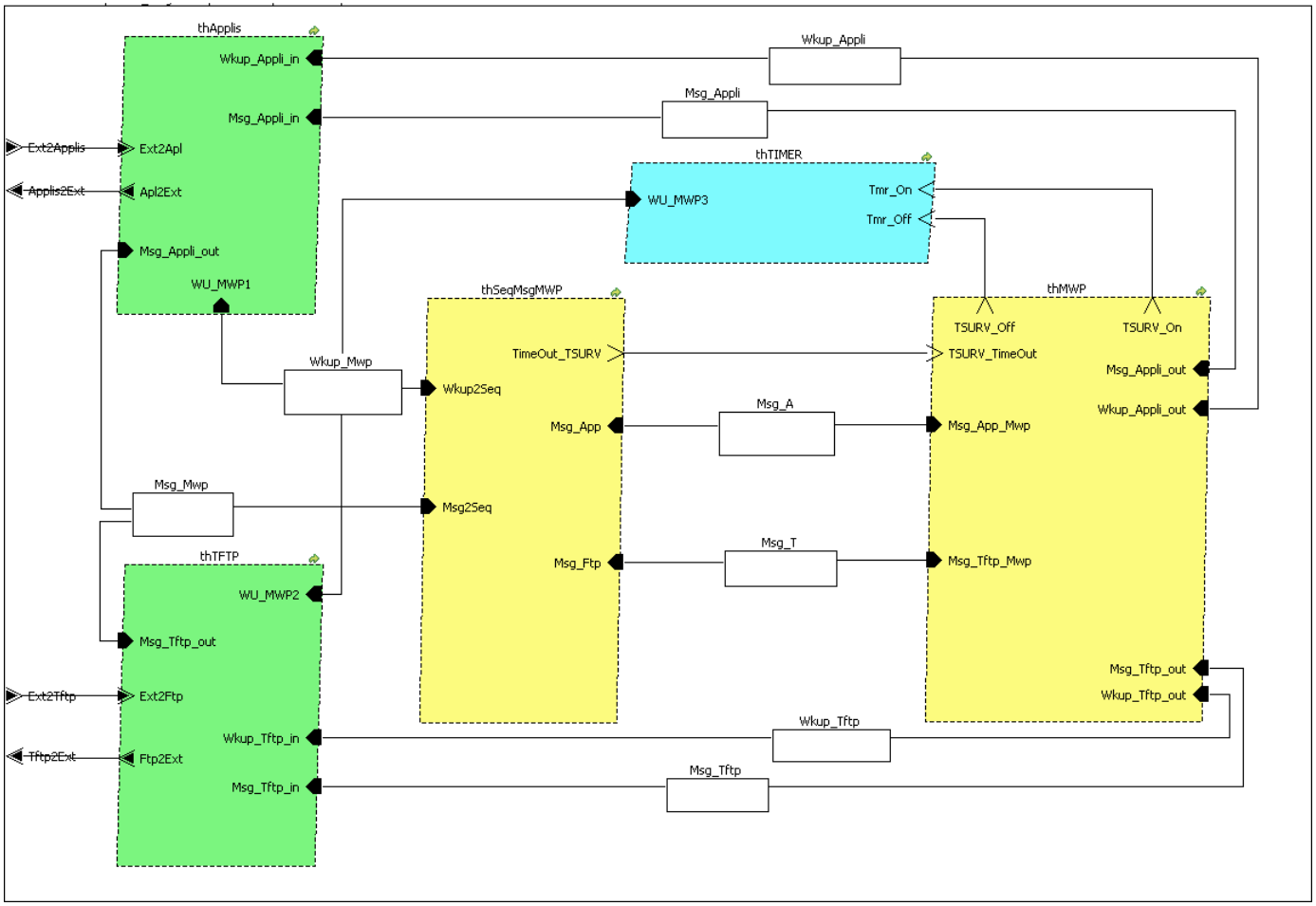
Air/Ground NPL







AADL model


thApplis
Wkup_Appli_in
Msg_Appli_in
Ext2Applis
Ext2Apl
Applis2Ext
Apl2Ext
Msg_Appli_out
WU_MWP1
Wkup_Appli
Msg_Appli
thTIMER
WU_MWP3
Tmr_On
Tmr_Off
thSeqMsgMWP
TimeOut_TSURV
Wkup2Seq
Msg_App
Msg2Seq
Msg_Ftp
thMWP
TSURV_Off
TSURV_On
TSURV_TimeOut
Msg_Appli_out
Wkup_Appli_out
Msg_App_Mwp
Msg_Tftp_Mwp
Msg_Tftp_out
Wkup_Tftp_out
Wkup_Mwp
Msg_A
Msg_Mwp
Msg_T
thTFTP
WU_MWP2
Msg_Tftp_out
Ext2Tftp
Ext2Ftp
Wkup_Tftp_in
Tftp2Ext
Ftp2Ext
Msg_Tftp_in
Wkup_Tftp
Msg_Tftp

Behavioral Description

```
thread tsend
features
  d: out event data port V.i;
  a: in event data port behavior::boolean
    {Queue_Size => 2; Dequeue_Protocol => OneItem;
     Overflow_Handling_Protocol => DropNewest;}; {...}
thread implementation tsend.i
subcomponents
    v1: data V.i;
annex behavior_specification {**
states
    s1: initial complete state;
    s2: complete state;
transitions
    s1 -[on a'count=0        ]-> s1 {v1.b := true; d!(v1);};
    s1 -[on a'fresh and a    ]-> s2 {computation(1ms,2ms);};
    s1 -[on a'fresh and not a]-> s1;
    s2 -[on a'count=0        ]-> s2 {v1.b := false; d!(v1);};
    s2 -[on a'fresh and not a]-> s1 {delay(1ms,2ms);};
    s2 -[on a'fresh and a    ]-> s2;
**};
end tsend.i;
{...}
thread implementation trecv.i
annex behavior_specification {**
transitions
    st -[d?(v) when v.b    ]-> sf {a!(v.b);};
    st -[d?(v) when not v.b]-> st {a!(v.b);};
    st -[on timeout 5 ms   ]-> st {a!(false);};
    sf -[d?(v) when not v.b]-> st {a!(false);};
    sf -[d?(v) when v.b    ]-> sf {a!(true);};
    sf -[on timeout 5 ms   ]-> sf {a!(true);};
**};
end trecv.i;
```

Experimental Results

- A “(small but) realistic model” that includes several threads, exchanging information through shared memory data
  - mainly message-passing
  - inherently asynchronous
  - the design of the communication architecture is prone to concurrency access problems

Experimental Results

- generation of the complete state space

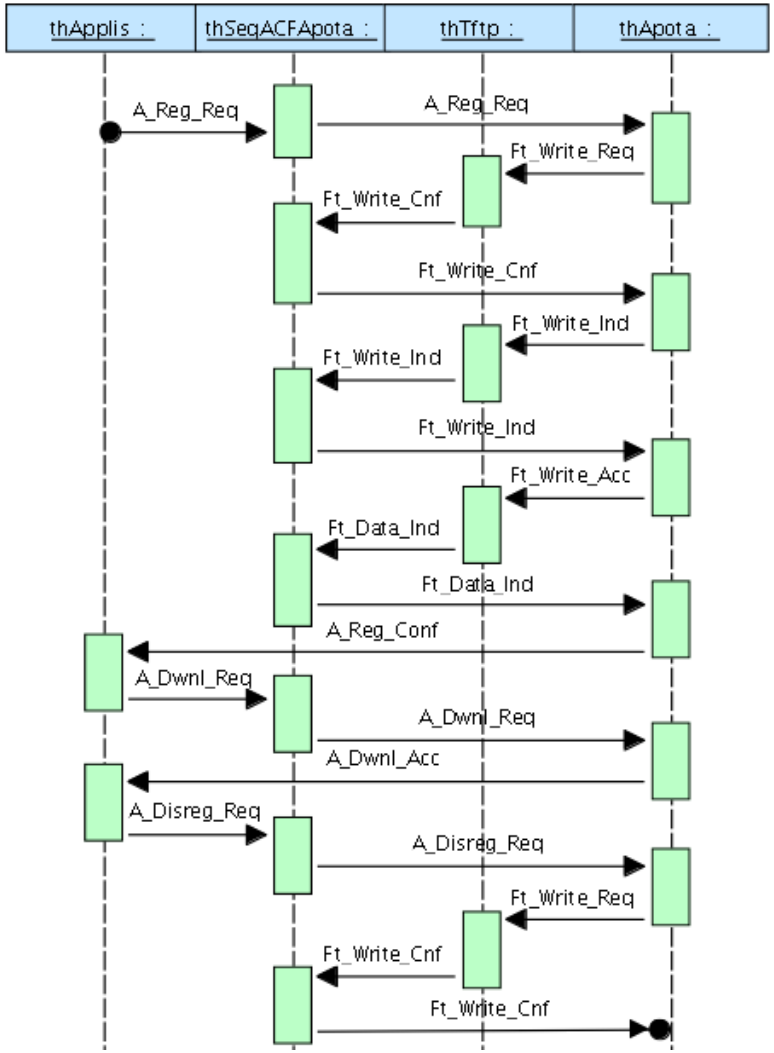
  $\sim 10^6$ states and 5.10^6 transitions (500 Mo memory footprint)

  $\sim$ 5 min. computation time on a 2 GHz proc. with 2 Go of RAM memory

Experimental Results

- Model-checking is able to check properties in a few seconds
  - 22 properties generated from the verification patterns → 2 minutes
  - 1 timed property (*… leadsto … within*)
  - 1 property corresponds to a sequence diagram

Experimental Results



CONCLUSION

Ideas for Breakout

- Where and how to add information about extended/behavioral requirements
  - e.g. links to sequence diagrams ; . . .

- What could be a “unit testing” or probes annex !

Ideas for Breakout (cont.)

- Contract-based modeling
  - as extra documentation ;
  - as an help for (compositional) formal verification and for simplifying the interpretation of models

Links

- http://projects.laas.fr/fiacre/

- Past projects :
  - ANR OpenEmbeDD: http://openembedd.org
  - ITEA Spices: http://www.spices-itea.org
  - ANR Itemis : http://itemis-anr.org
  - FNRAE Quarteft: http://quarteft.loria.fr
  - ARTEMIS Cesar : http://cesarproject.eu/
- Current projects
  - ITEA2 openETCS
  - FUI Projet P