



Prototyping an Embedded Automotive System from its UML/SysML Models

Ludovic Apvrille, Alexandre Becoulet

Institut Telecom, Telecom ParisTech, CNRS LTCI
ludovic.apvrille@telecom-paristech.fr
alexandre.becoulet@telecom-paristech.fr

ERTS² 2012, Feb 1st, Toulouse

Prototyping

Context

AVATAR

SoCLib, MutekH

Contribution

Overall approach

Case study

Applying the methodology

Conclusion

Context

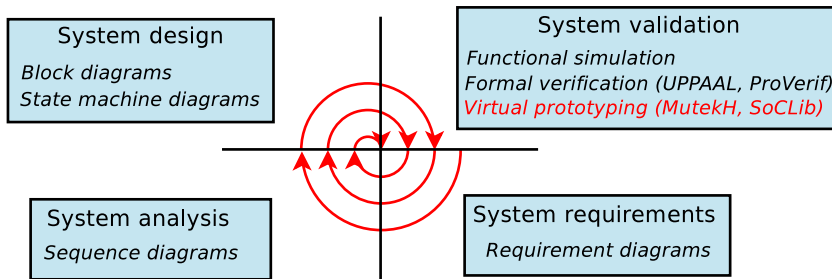
Embedded systems prototyping

- ▶ = Testing in "quite" realistic conditions SW before the HW platform is available
- ▶ Cumbersome task ⇒ We try to ease this process!

Our approach is based on the AVATAR environment

- ▶ Based on high-level models (UML / SysML)
- ▶ Supported with a free software: TTool
 - ▶ Modeling, functional simulation
 - ▶ Formal proof
 - ▶ Model-checking, SAT solving
 - ▶ **New:** Virtual prototyping
 - ▶ Based on SoCLib (target platform) and MutekH (Operating System)

AVATAR Methodology



SoCLib and MutekH

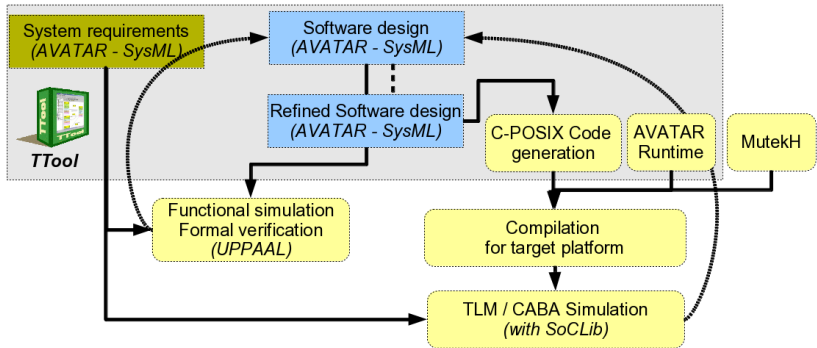
Hardware platform simulator: SoCLib

- ▶ Virtual prototyping of complex Systems-on-Chip
- ▶ Supports several models of processors, buses, memories
 - ▶ Example of CPUs: MIPS, ARM, SPARC, Nios2, PowerPC
- ▶ Two sets of simulation models:
 - ▶ TLM = Transaction Level Modeling
 - ▶ CABA = Cycle Accurate Bit Accurate

Embedded Operating System: MutekH

- ▶ Natively handles heterogeneous multiprocessor platforms
- ▶ POSIX threads support
- ▶ Note: any Operating System supporting POSIX threading and that can be compiled for SoCLib could be used

AVATAR Methodology



Case Study: An Automotive Application

Automotive embedded system

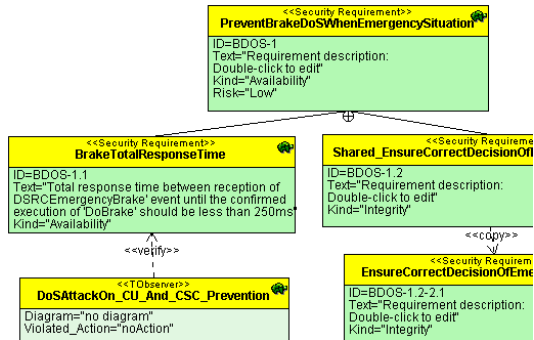
- ▶ Made upon a hundred of Electronic Control Units (ECUs)
- ▶ Interconnection with CAN / FlexRay / MOST

Automatic Braking Application

- ▶ Taken from Intelligent Transport System applications and from the EVITA european project
1. An obstacle is detected by a car
 2. That information is broadcasted to neighborhood cars
 3. A car receiving such an information may decide to make an automatic emergency braking w.r.t.:
 - ▶ Vehicle dynamics, vehicle position, ... → Plausibility check

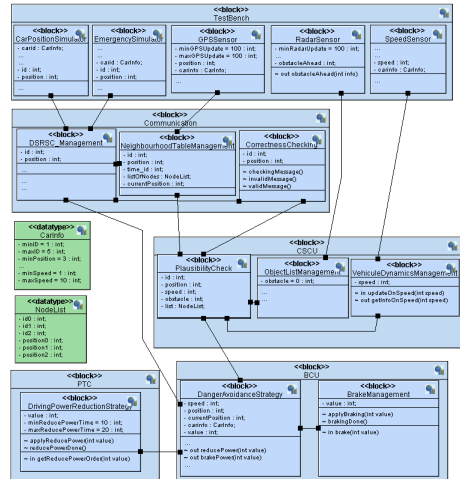
Requirement Capture

- ▶ Based on SysML requirement diagrams
- ▶ Safety and security related requirements
- ▶ Observers

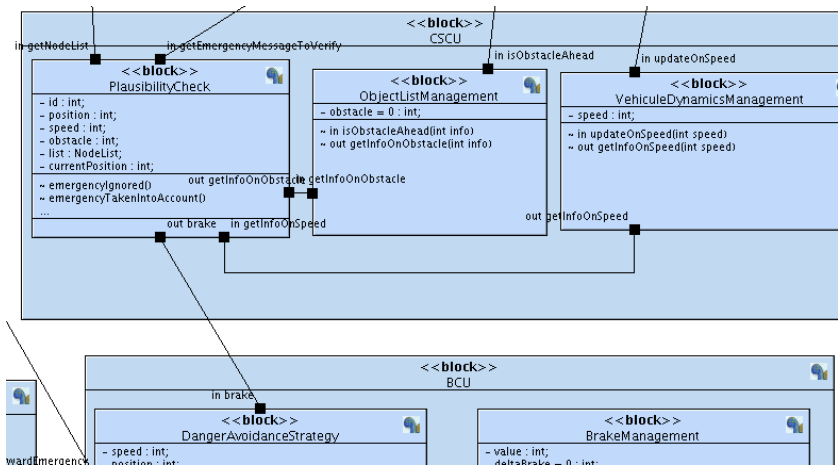


Design: System Architecture

- ▶ Architecture is described with SysML block diagrams
- ▶ AVATAR block = id + attributes + methods + in/out signals
- ▶ Asynchronous or synchronous communications with signals
- ▶ Closed system i.e. use cases of the system have to be described in the model as well

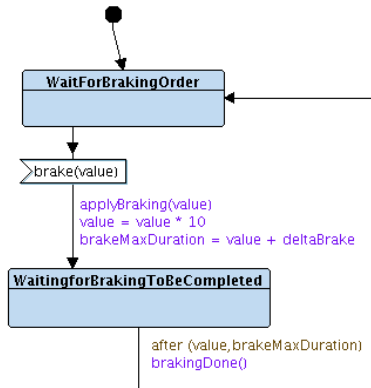


Design : System Architecture (Cont.)

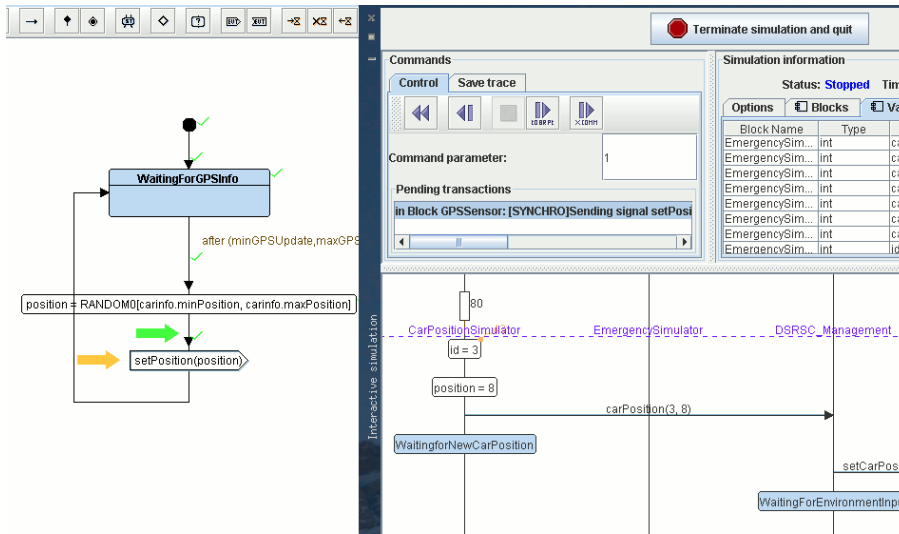


Design: System Behavior

- ▶ Behavior is described in SysML state machine diagrams
- ▶ All usual logical actions (variable modifications, method calls, etc.)
- ▶ Signal sending / receiving
- ▶ Nested states
- ▶ Temporal intervals
 - ▶ *After*(t_{min} , t_{max})
 - ▶ *ComputeFor*(t_{min} , t_{max})

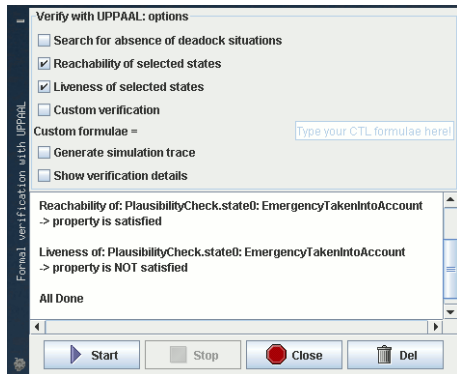


Functional Interactive Simulation (Zoom)



Formal Verification

- ▶ Press-button approach
 - ▶ Safety properties (reachability, liveness)
 - ▶ Security properties (confidentiality, authenticity)
 - ▶ Based on UPPAAL, ProVerif
- ▶ Example
 - ▶ Prove whether the car may make an emergency braking, or not
 - ▶ i.e., is the *EmergencyTakenIntoAccount* state reachable in the block *PlausibilityCheck*?



Formal Verification (Cont.)

Formal verification with UPPAAL

Verify with UPPAAL: options

- Search for absence of deadlock situations
- Reachability of selected states
- Liveness of selected states
- Custom verification

Custom formulae =

- Generate simulation trace
- Show verification details

Reachability of: PlausibilityCheck.state0: EmergencyTakenIntoAccount
 -> property is satisfied

Liveness of: PlausibilityCheck.state0: EmergencyTakenIntoAccount
 -> property is NOT satisfied

All Done

Prototyping Steps

1. Model refinement
2. Selection of an OS, setting of options of this OS (scheduling algorithm, ...)
3. Selection of a hardware platform, and selection of a task allocation scheme
4. Code generation (press-button approach)
5. Manual code improvement
6. Code compilation and linkage with OS
7. Simulation platform boots the OS and executes the code
8. Execution analysis: directly in TTool (sequence diagram) or with debuggers (e.g., gdb)

Prototyping: Graphical Environment

Main window of TTool

Code generation window

Console of MutekH

UML sequence diagram updated when simulating with SoCLib

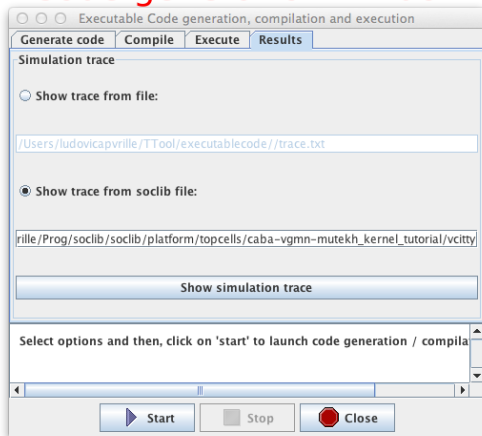
SoCLib simulation based on a SystemC engine

SystemCAS

347 transactions, min time=2, max time=57001

Prototyping: Code Generation

Code generation window



Prototyping: SocLib Simulation

The screenshot shows a terminal window with the following content:

```

echo "running soclib"
running soclib
cd ~/Prog/soclib/soclib/platform/topcells/caba-vgmn-mutekh_kernel_tutorial; SOCLIB_GDB=S ./system.x ppc405:5 ~/Prog/mutekh/avatar-soclib-ppc.out

```

Below the terminal output, the text "SoCLib simulation based on a SystemC engine" is overlaid in red. Underneath this, the "SYSTEMCASS" logo is displayed in a stylized green font. Below the logo, the following text is shown:

```

Cycle Accurate System Simulator
ASIM/LIP6/UPMC
E-mail support: Richard.Buchmann@asim.lip6.fr
Contributors : Richard Buchmann, Sami Taktak,
Paul-Jerome Kingbo, Frederic P?trot,
Nicolas Pouillon

Last change : Dec 6 2011

Initializing memories with 5a
caba-vgmn-mutekh_kernel_tutorial SoCLib simulator for MutekH
Initializing memories with 5a
Initializing memories with 5a

```

Prototyping: Console

```

vcitty
-> Locking mutex
DT> Adding pending request in inWaitqueue
DT - #629 time=75.998696832 block=NeighbourhoodTableManagement type=state_entering
state=waitingForNewNodesOrPosition
VehiculeDynamicsManagement -> Waiting for request!

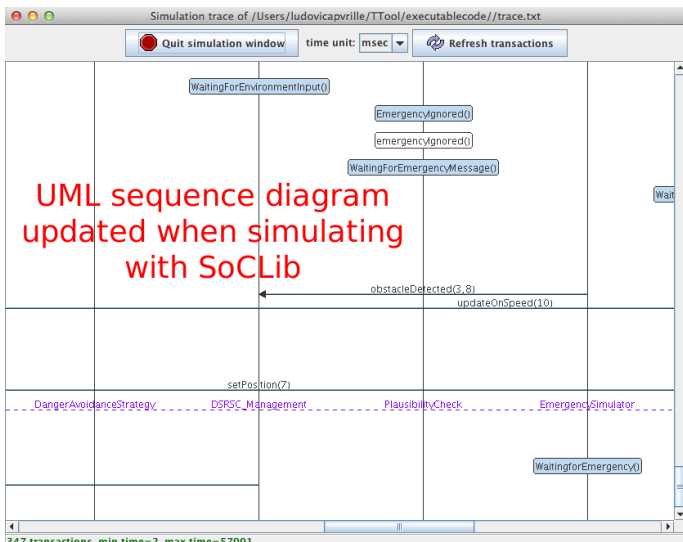
DT - VehiculeDynamicsManagement -> DT - Releasing mutexLocking mutex

DT - ObjectListManagement -> Mutex locked
DT - ObjectListManagement -> Going to execute request
DT> No request selected -> looking for one!
DT> Counting requests
DT> Starting loop
DT> receive sync
DT> Send sync
DT> Send sync not executable
DT> Counting requests=: 0
DT> No pending requests
DT> Adding pending request in inWaitqueue
DT> Adding pending request in outWaitqueue
DT - ObjectListManagement -> Waiting for request!
DT - DT - ObjectListManagement -> Releasing mutexLocking mutex

[]
  
```

Console of MutekH

Prototyping: Trace



Conclusion and Future Work

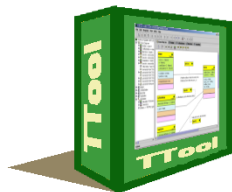
Prototyping environment

- ▶ High-level models
- ▶ Press-button approach
 - ▶ Simulation, formal verification, prototyping
- ▶ Fully based on free software (TTool, SoCLib, MutekH)

Future work

- ▶ Integration of C-code directly in the AVATAR model
- ▶ Optimization of the AVATAR-to-C code generator
- ▶ Better usage of SoCLib / MutekH capabilities
- ▶ Animation of State Machines at prototyping phase

Questions?



ttool.telecom-paristech.fr

www.soclib.fr

www.mutekh.fr