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

ERTS^2 2012, Feb 1^{st}, Toulouse

Outline

Prototyping
- Context
- AVATAR
- SoCLib, MutekH

Contribution
- Overall approach
- Case study
- Applying the methodology

Conclusion

Context

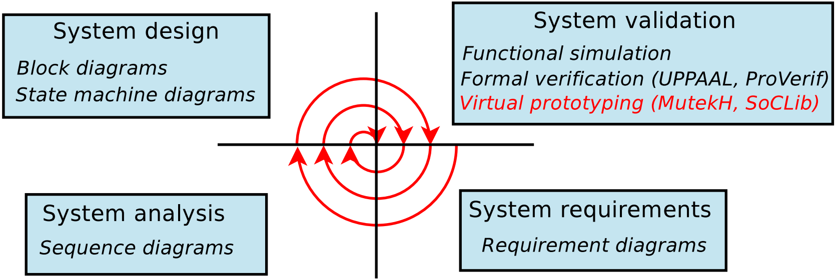
Embedded systems prototyping

- = 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

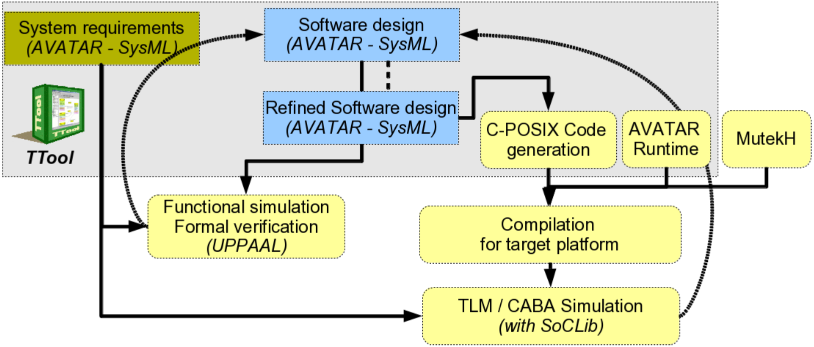
Hardware platform simulator: SoCLib

- 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

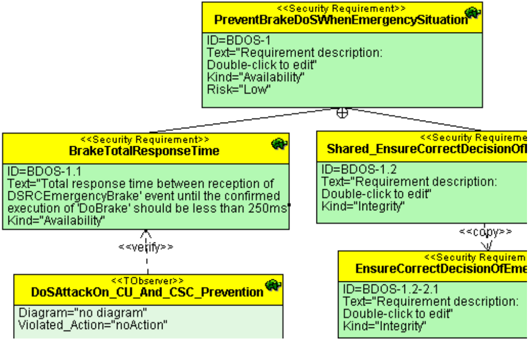
Automatic Braking Application

- 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

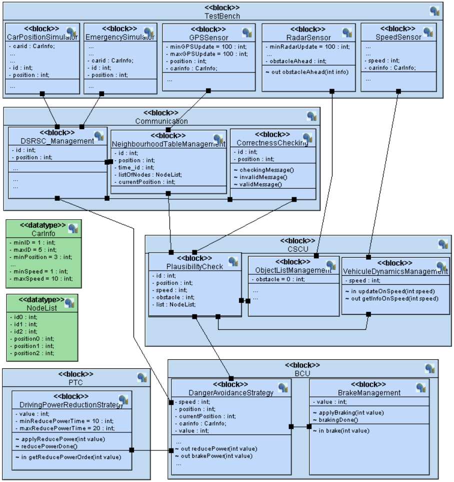
Requirement Capture

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

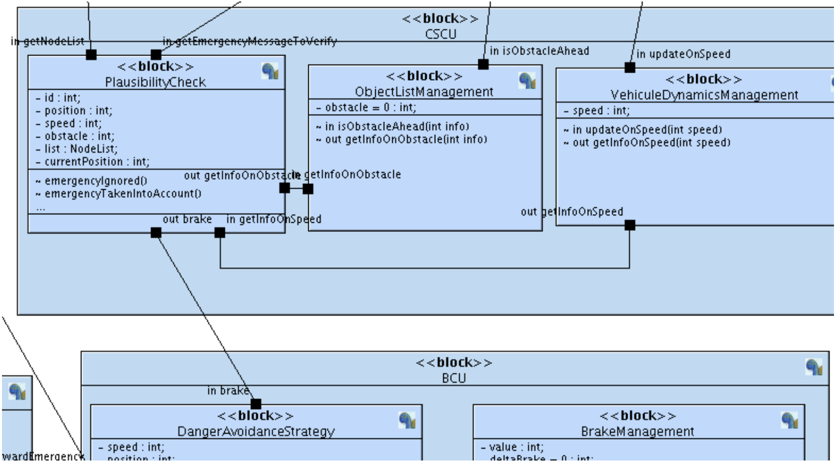


Design: System Architecture

- 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

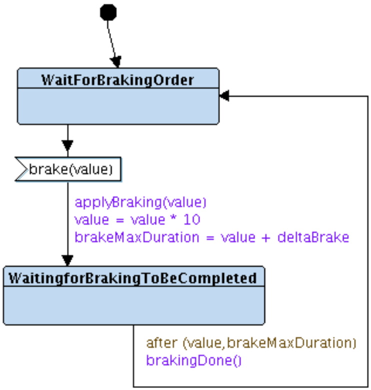


Design : System Architecture (Cont.)



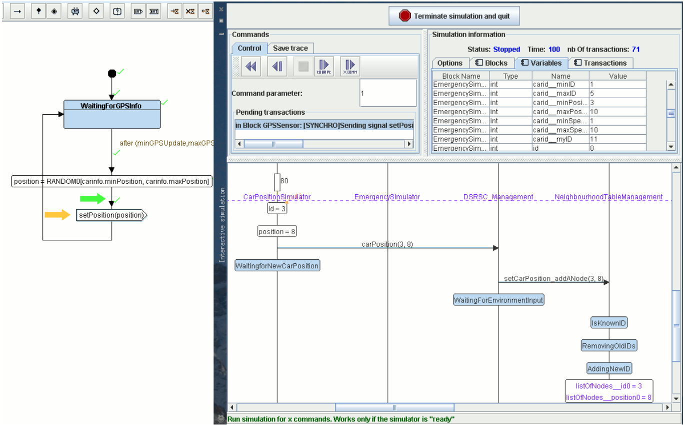
Design: System Behavior

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

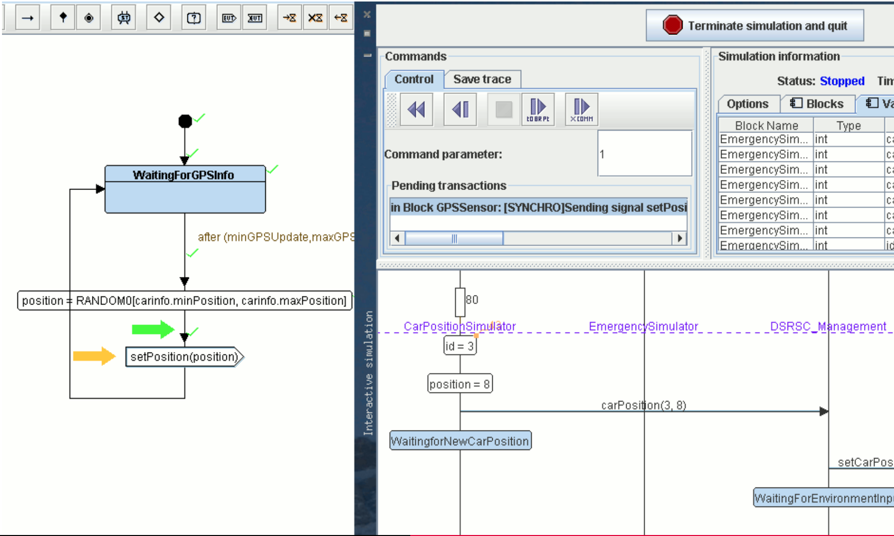


Functional Interactive Simulation

- Executed by TTool
- Assumes no hardware and no OS: interactive execution of state machines
- Traces = Sequence diagram
- Model animation

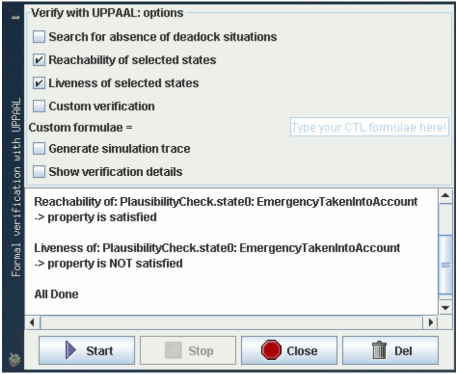


Functional Interactive Simulation (Zoom)

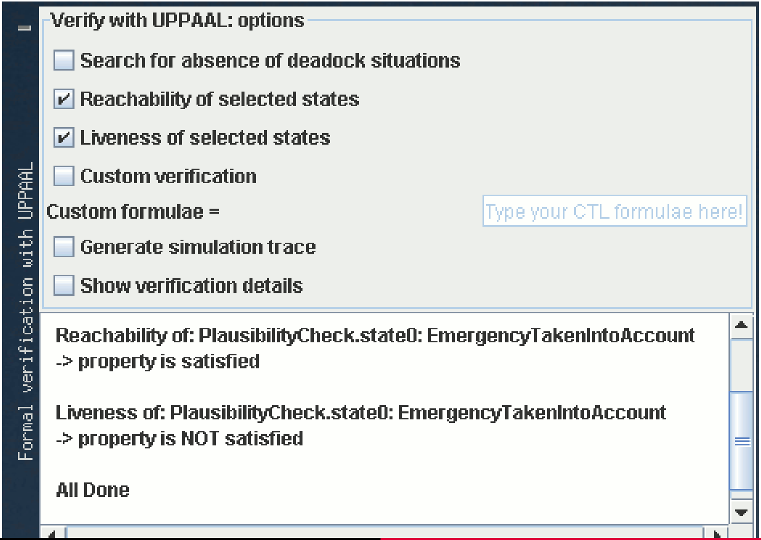


Formal Verification

- 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*?



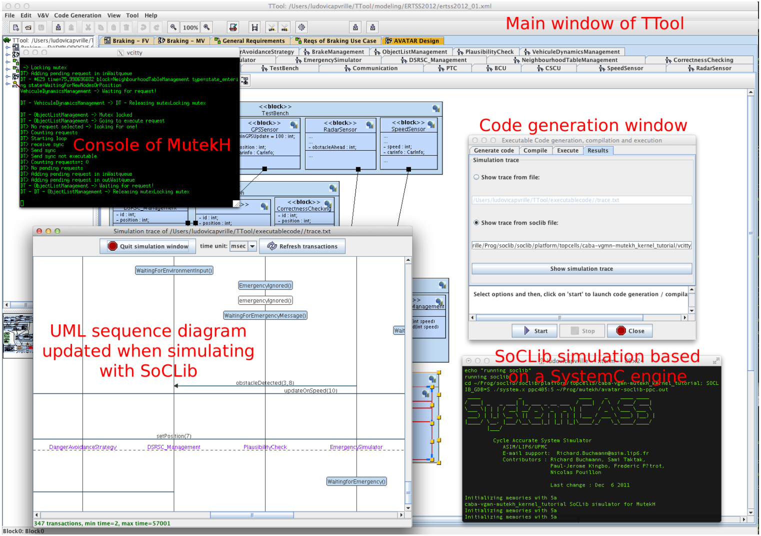
Formal Verification (Cont.)



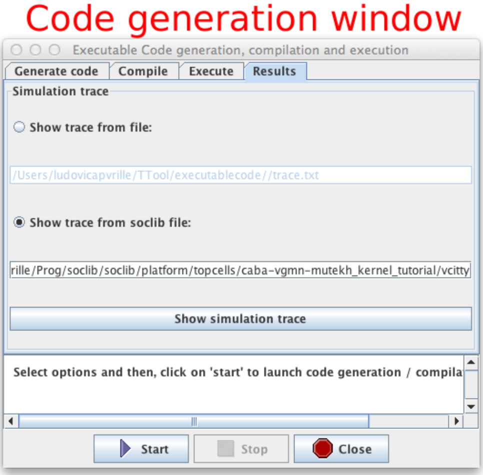
Prototyping Steps

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)

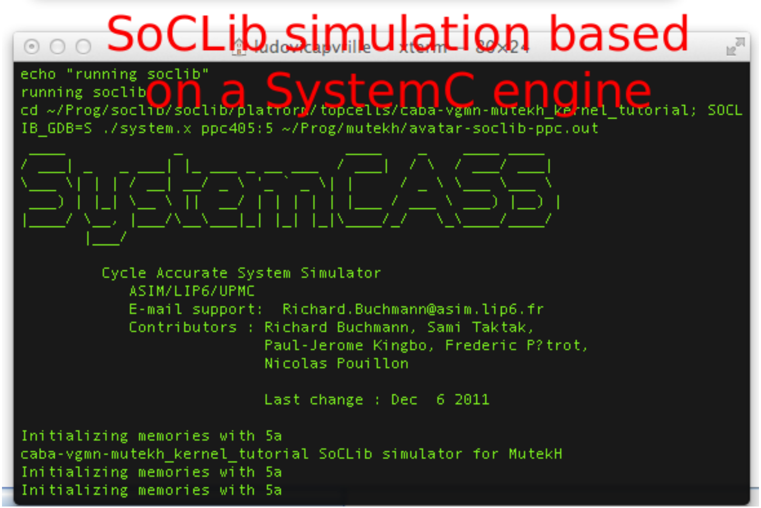
Prototyping: Graphical Environment



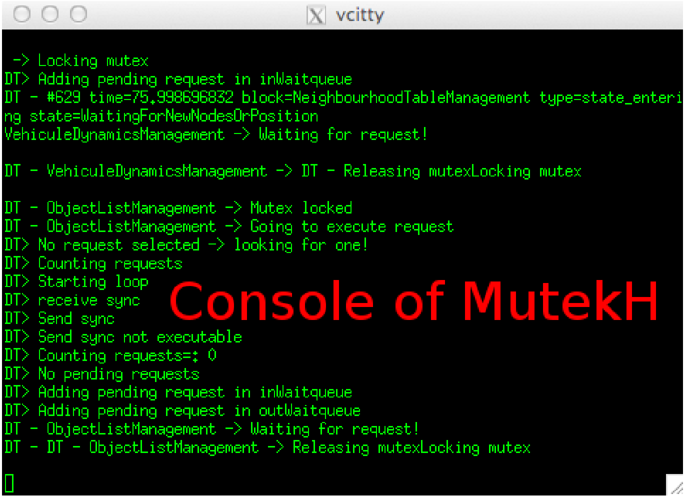
Prototyping: Code Generation



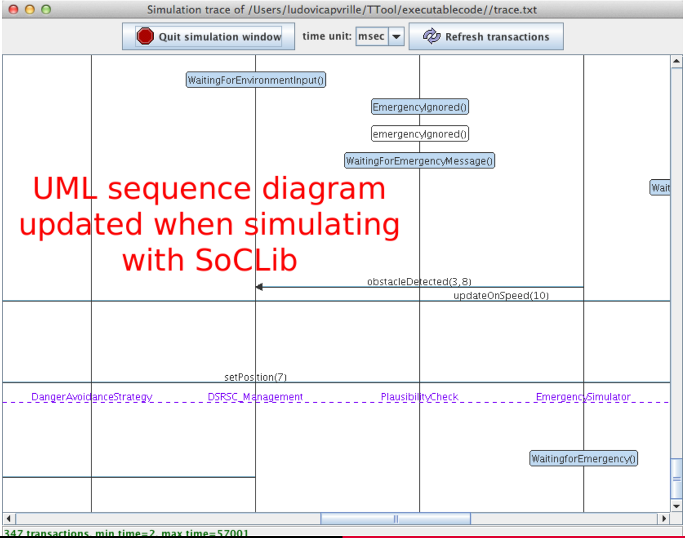
Prototyping: SocLib Simulation



Prototyping: Console



Prototyping: Trace



Conclusion and Future Work

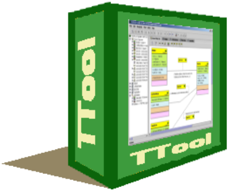
Prototyping environment

- 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