Institut
Mines-Telecom

Virtual Prototyping from SysML Models

Presentation and Demonstrations

Ludovic Apvrille
ludovic.apvrille@telecom-paristech.fr

Journée SysML France

Outline

Outline

Definition

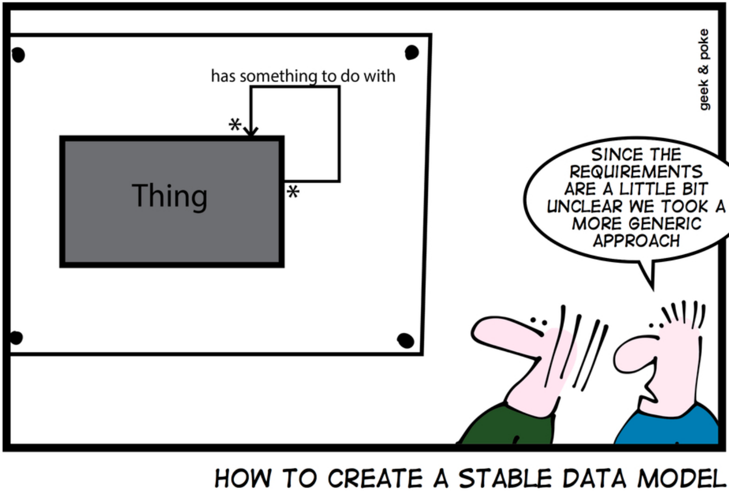
Generic definition

- From greek: $\pi\rho\omega\tau o\tau\upsilon\tau o\upsilon$
- **A prototype is an early sample, model or release of a product built to test a concept or process or to act as a thing to be replicated or learned from** (Wikipedia)

In our context

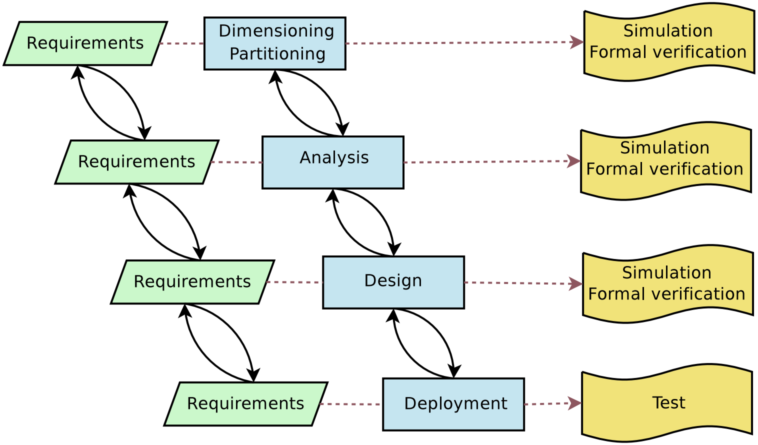
- Prototyping = experimenting a functional model mapped onto a concrete hardware architecture
- Virtual prototyping = experimenting a functional model mapped onto a hardware model
  - Hardware model can be more or less abstract/concrete
    - Example : CPUs emulated with Instruction Set Simulator, abstract instructions, etc.

Abstraction Level


has something to do with
*
*
Thing
geek & poke
SINCE THE REQUIREMENTS ARE A LITTLE BIT UNCLEAR WE TOOK A MORE GENERIC APPROACH
HOW TO CREATE A STABLE DATA MODEL

(*Source: Peek and Poke, July, 2013*)

Developing an Embedded System: Methodology and Abstraction levels


Requirements
Dimensioning Partitioning
Simulation Formal verification
Requirements
Analysis
Simulation Formal verification
Requirements
Design
Simulation Formal verification
Requirements
Deployment
Test

Virtual Prototyping at Two Abstraction Levels

Partitioning

- Hardware highly abstracted
- Goal: Analyzing various functionally equivalent implementation alternatives
  - Mapping of functions and communications
- Metrics:
  - CPU load
  - Bus occupation
  - Silicon area

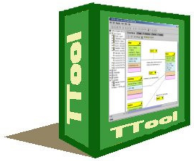
Deployment

- Functions to be software-implemented have been designed
- Hardware target is not yet available
  - Or not easy/practical to use
- Goal: Determining precise timing and control issues
  - Scheduling policy
  - Latencies
  - Missed deadlines

TTool: A Multi Profile Platform

TTool

- Open-source toolkit mainly developed by Telecom ParisTech / COMELEC
- Multi-profile toolkit
  - DIPLODOCUS, AVATAR, ...
- Support from academic (e.g. INRIA, ISAE) and industrial partners (e.g., Freescale)



Main ideas

Lightweight, easy-to-use toolkit

Simulation with model animation

Formal proof at the push of a button

Outline

DIPLODOCUS in a Nutshell

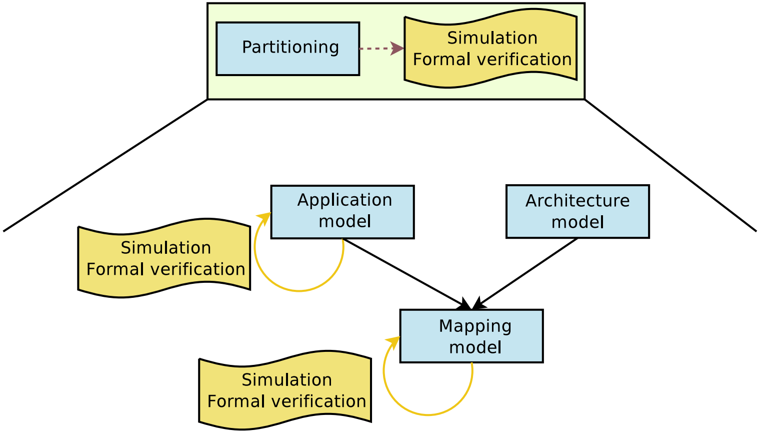
DIPLODOCUS = UML Profile

- System-level Design Space Exploration
- Y-Methodology

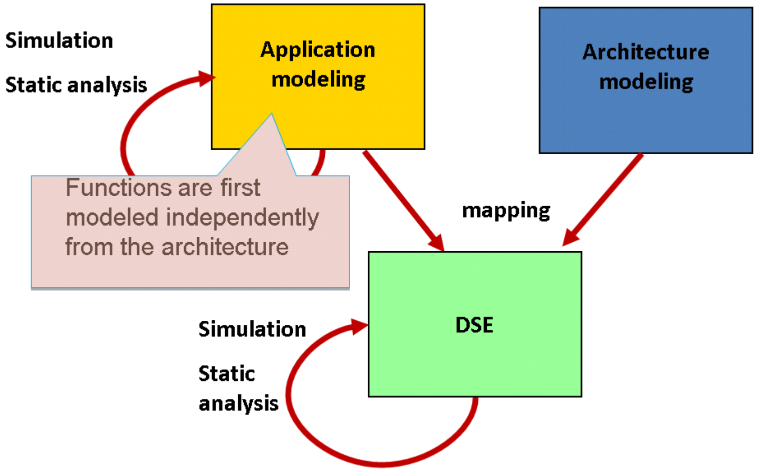
Main features

- High level of abstraction
  - Exchanged data are unvalued
  - Complexity operator
  - Parametrized hardware nodes
- Formal semantics
- Very fast simulation support

Partitioning with the Y-Methodology

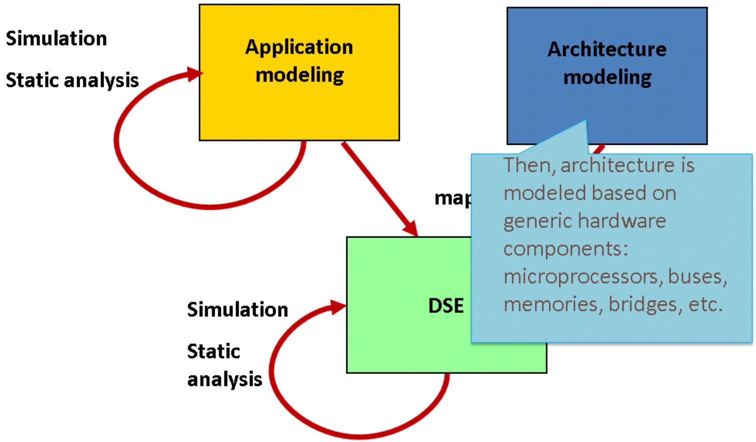


Application Modeling



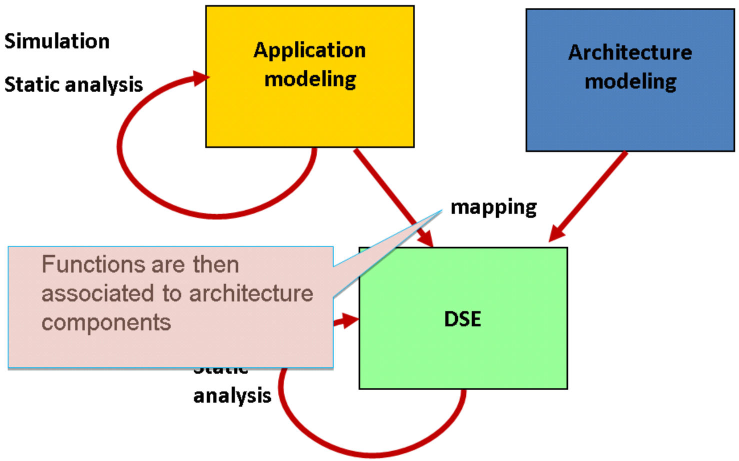
Functions are first modeled independently from the architecture

Architecture Modeling



Then, architecture is modeled based on generic hardware components: microprocessors, buses, memories, bridges, etc.

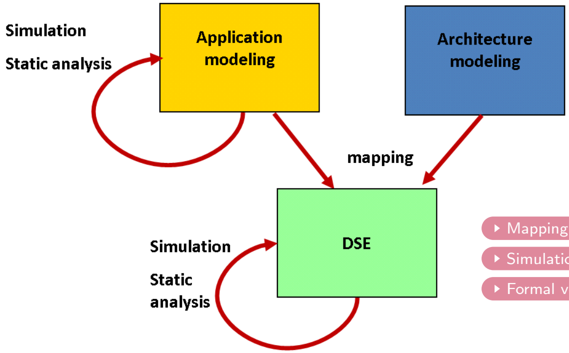
Mapping



Functions are then associated to architecture components

Browsing the DIPLODOCUS Methododology

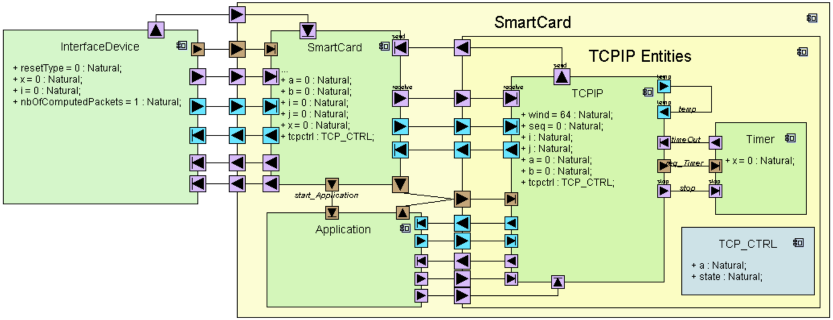
▸ Application structure
▸ Application behavior
▸ Formal verification



▸ Architecture model

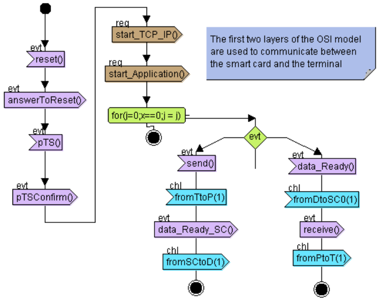
▸ Mapping model
▸ Simulation
▸ Formal verification

Application Structure (Smart Card system)



Back to methodology

Application Behavior

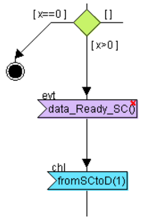


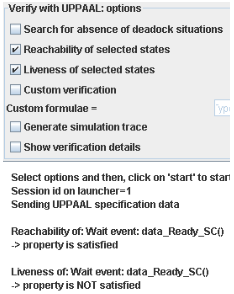
Activity Diagram of the SmartCard component

Back to methodology

Formal Verification at Application Level

- No assumption on the underlying architecture
- All possible interleavings between actions are considered
- Formal verification is based on LOTOS/CADP or UPPAAL
    - Press-button approach




Verify with UPPAAL: options

- [ ] Search for absence of deadock situations
- [x] Reachability of selected states
- [x] Liveness of selected states
- [ ] Custom verification

Custom formulae =

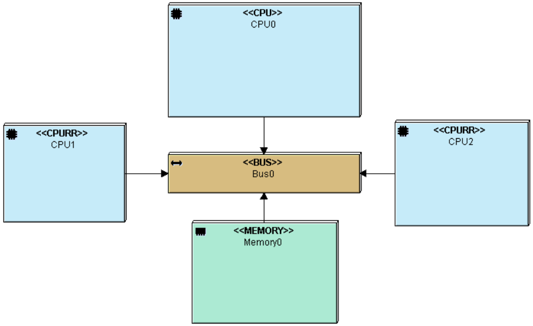
- [ ] Generate simulation trace
- [ ] Show verification details

```
Select options and then, click on 'start' to start
Session id on launcher=1
Sending UPPAAL specification data

Reachability of: Wait event: data_Ready_SC()
-> property is satisfied

Liveness of: Wait event: data_Ready_SC()
-> property is NOT satisfied
```

Back to methodology

Architecture

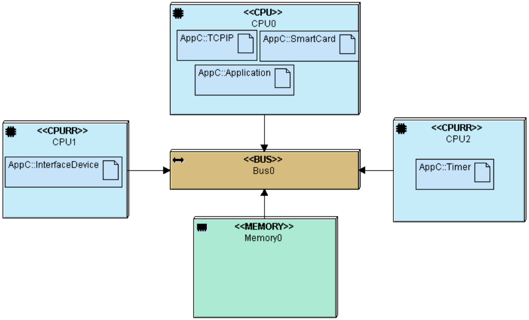
- Given in terms of parameterized nodes
- CPU, HWA, Bus, Memory, Bridge, etc.
- CPU parameters: scheduling policy, cache miss ratio, miss-branching prediction, pipeline size, etc.



Back to methodology

Mapping

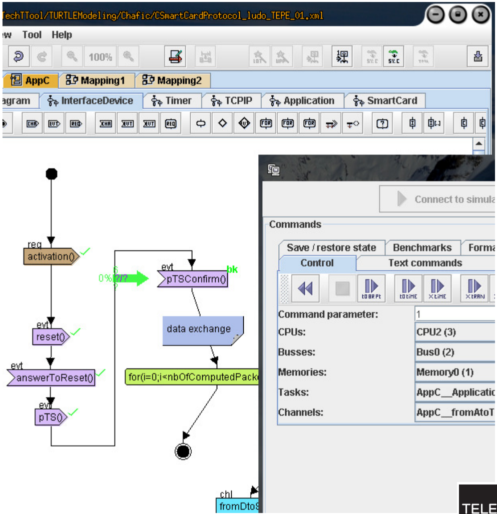
- Task are mapped on execution nodes (e.g., CPUs, HWAs)
- Channels are mapped on communication and storage nodes



Back to methodology

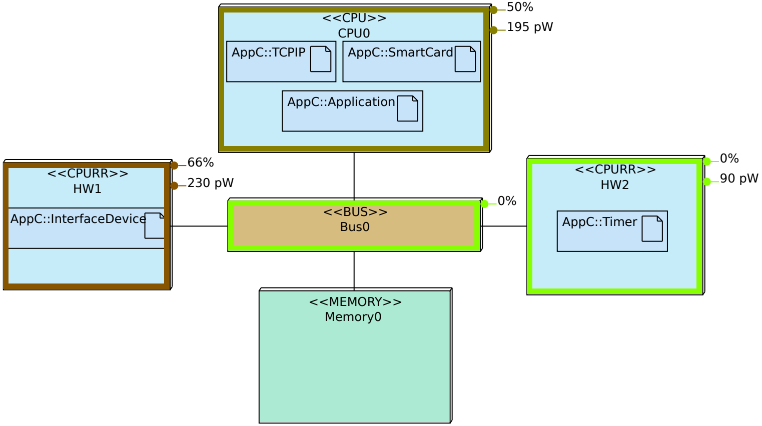
After-Mapping Simulation

- TTool Built-in simulator
- **Extremely fast**
- Diagram animation
- Step-by-step execution, breakpoints, etc.



Command parameter:	1
CPUs:	CPU2 (3)
Busses:	Bus0 (2)
Memories:	Memory0 (1)
Tasks:	AppC__Applicatio
Channels:	AppC__fromAtoT

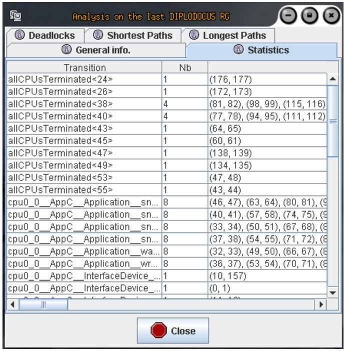
After-Mapping Simulation (Cont.)



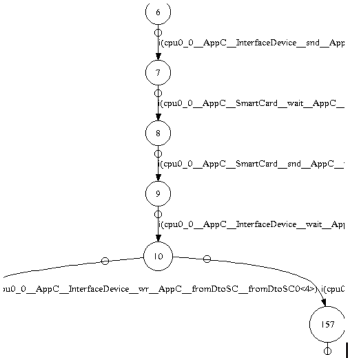
Back to methodology

After-Mapping Formal Verification

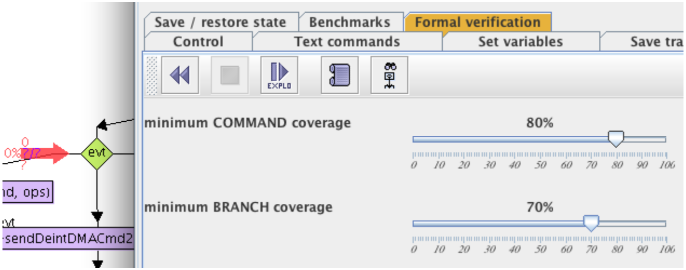
- TTool built-in simulator can compute all possible execution paths
- Graph analysis and visualization



Transition	Nb	
allCPUsTerminated<24>	1	(176, 177)
allCPUsTerminated<26>	1	(172, 173)
allCPUsTerminated<38>	4	(81, 82), (98, 99), (115, 116)
allCPUsTerminated<40>	4	(77, 78), (94, 95), (111, 112)
allCPUsTerminated<43>	1	(64, 65)
allCPUsTerminated<45>	1	(60, 61)
allCPUsTerminated<47>	1	(138, 139)
allCPUsTerminated<49>	1	(134, 135)
allCPUsTerminated<53>	1	(47, 48)
allCPUsTerminated<55>	1	(43, 44)
cpu0_0__AppC__Application__sn...	8	(46, 47), (63, 64), (80, 81), (9
cpu0_0__AppC__Application__sn...	8	(40, 41), (57, 58), (74, 75), (9
cpu0_0__AppC__Application__sn...	8	(33, 34), (50, 51), (67, 68), (8
cpu0_0__AppC__Application__sn...	8	(37, 38), (54, 55), (71, 72), (8
cpu0_0__AppC__Application__wa...	8	(32, 33), (49, 50), (66, 67), (8
cpu0_0__AppC__Application__wr...	8	(36, 37), (53, 54), (70, 71), (8
cpu0_0__AppC__InterfaceDevice_...	1	(10, 157)
cpu0_0__AppC__InterfaceDevice_...	1	(0, 1)


6
i(cpu0_0__AppC__InterfaceDevice__snd__App
7
i(cpu0_0__AppC__SmartCard__wait__AppC__
8
i(cpu0_0__AppC__SmartCard__snd__AppC__
9
i(cpu0_0__AppC__InterfaceDevice__wait__Ap
10
u0_0__AppC__InterfaceDevice__wr__AppC__fromDtoSC__fromDtoSC0<4>)
i(cpu
157

After-Mapping Coverage-Enhanced Simulation



minimum COMMAND coverage 80%

minimum BRANCH coverage 70%

Possibility to select a given part of the model to be explored

- Minimum percentage of operators coverage
- Minimum percentage of branch coverage

Implementation: TTool built-in model-checking techniques

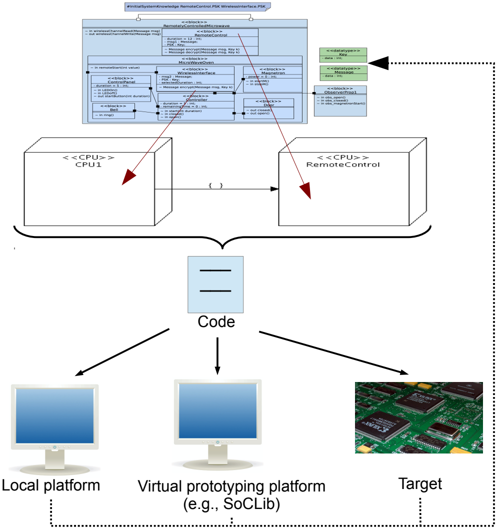
Back to methodology

A Few Case Studies ...

- MPEG coders and decoders (Texas Instruments)
- LTE SoC (Freescale)
- Partitioning in vehicle embedded systems (EVITA project)
- Partitioning and code generation for Software-Defined Radio systems (SACRA project)

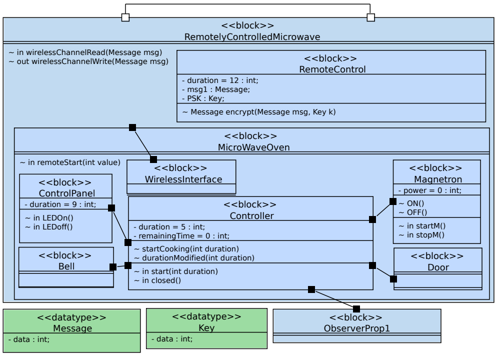
Outline

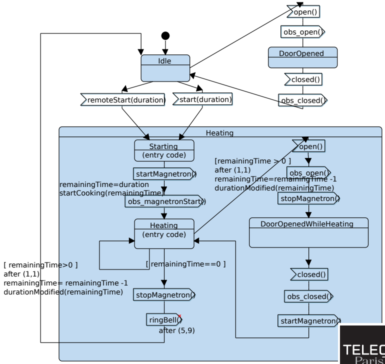
Deployment: Overview



AVATAR Design

- Block Definition and Internal Block diagrams are merged
- Synchronous and asynchronous communications
- Interactive simulation
- Formal verification of safety and security properties

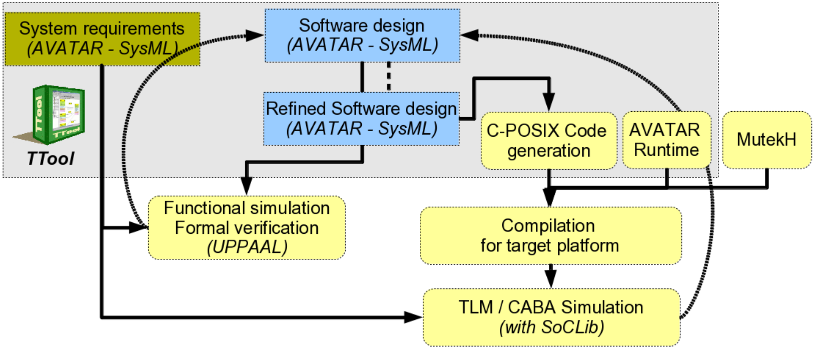




Principle of Code Generation

- Only AVATAR design diagrams are taken into account
- Generated code relies on POSIX threads
  - One thread per block
- Synchronous communications between blocks is implemented in the AVATAR runtime with POSIX mutex
  - Asynchronous communications relies on linked lists managed in the AVATAR runtime
  - Time is handled based on POSIX *clock_gettime*() with *CLOCK_REALTIME* option
  - ...

Method


System requirements
(AVATAR - SysML)
TTool
Software design
(AVATAR - SysML)
Refined Software design
(AVATAR - SysML)
C-POSIX Code generation
AVATAR Runtime
MutekH
Functional simulation
Formal verification
(UPPAAL)
Compilation
for target platform
TLM / CABA Simulation
(with SoCLib)

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 - Code might also be manually added at model level
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*)

Support: SoCLib and MutekH

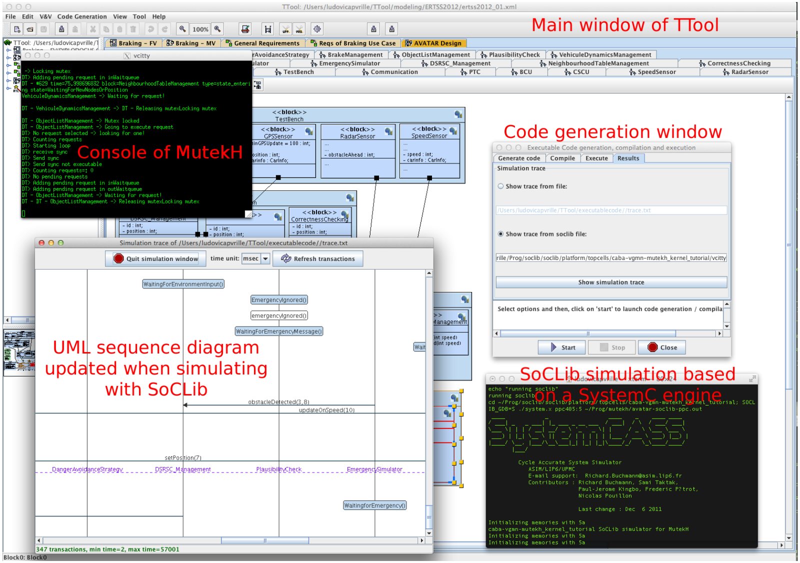
Hardware platform simulator: SoCLib (www.soclib.fr)

- 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 (www.mutekh.fr)

- 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

Graphical Environment


Main window of TTool
Console of MutekH
Code generation window
UML sequence diagram updated when simulating with SoCLib
SoCLib simulation based on a SystemC engine
TTool: /Users/ludovicapvrille/TTool/modeling/ERTSS2012/ertss2012_01.xml
File
Edit
V&V
Code Generation
View
Tool
Help
Braking – FV
Braking – MV
General Requirements
Reqs of Braking Use Case
AVATAR Design
vcitty
Simulation trace of /Users/ludovicapvrille/TTool/executablecode//trace.txt
Quit simulation window
time unit:
msec
Refresh transactions
Executable Code generation, compilation and execution
Generate code
Compile
Execute
Results
Start
Stop
Close

```
 -> Locking mutex
DT> Adding pending request in inWaitqueue
DT - #629 time=75.998696832 block=NeighbourhoodTableManagement type=state_enteri
ng 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
```

Simulation trace

○ Show trace from file:

/Users/ludovicapvrille/TTool/executablecode//trace.txt

● Show trace from soclib file:

rille/Prog/soclib/soclib/platform/topcells/caba-vgmn-mutekh_kernel_tutorial/vcitty

Show simulation trace

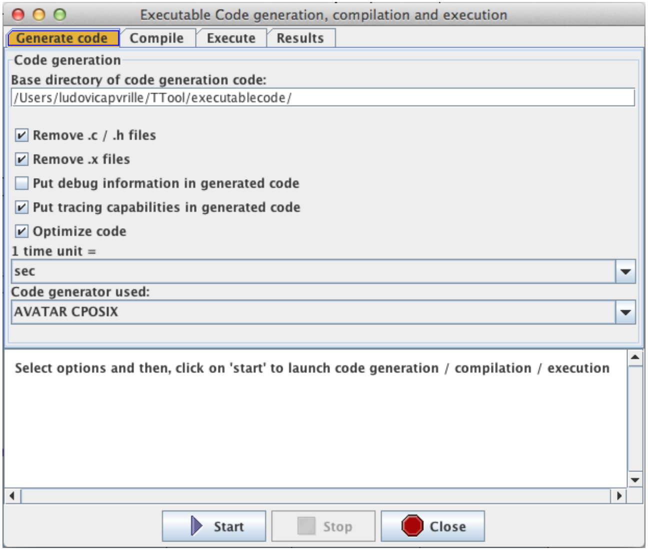
Select options and then, click on 'start' to launch code generation / compila

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

Block0: Block0

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

          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
```

(Virtual) Prototyping: Code Generation


Executable Code generation, compilation and execution

Generate code | Compile | Execute | Results

Code generation

Base directory of code generation code:
/Users/ludovicapvrille/TTool/executablecode/

- [✓] Remove .c / .h files
- [✓] Remove .x files
- [ ] Put debug information in generated code
- [✓] Put tracing capabilities in generated code
- [✓] Optimize code

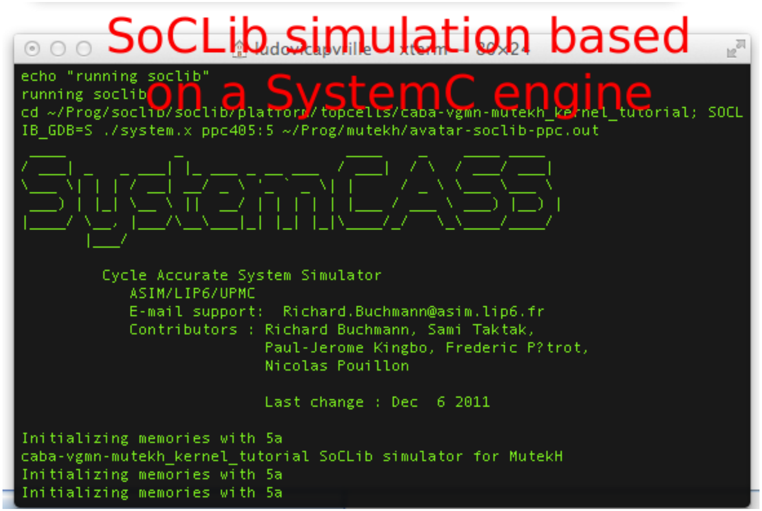
1 time unit =
sec

Code generator used:
AVATAR CPOSIX

Select options and then, click on 'start' to launch code generation / compilation / execution

Virtual Prototyping: SoCLib Simulation

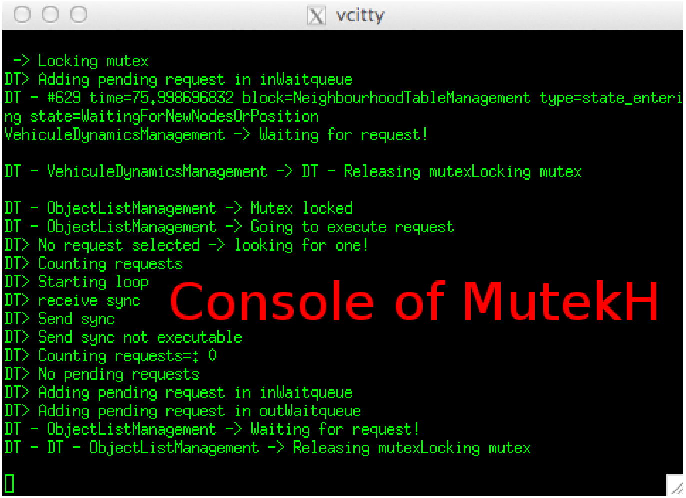
SoCLib simulation based on a SystemC engine



```
echo "running soclib"
running soclib
cd ~/Prog/soclib/soclib/platform/topcells/caba-vgmn-mutekh_kernel_tutorial; SOCL
IB_GDB=S ./system.x ppc405:5 ~/Prog/mutekh/avatar-soclib-ppc.out
 ____            _                  ____    _    ____ ____
/ ___| _   _ ___| |_ ___ _ __ ___  / ___|  / \  / ___/ ___|
\___ \| | | / __| __/ _ \ '_ ` _ \| |     / _ \ \___ \___ \
 ___) | |_| \__ \ ||  __/ | | | | | |___ / ___ \ ___) |__) |
|____/ \__, |___/\__\___|_| |_| |_|\____/_/   \_\____/____/
       |___/

        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
```

TELECOM ParisTech

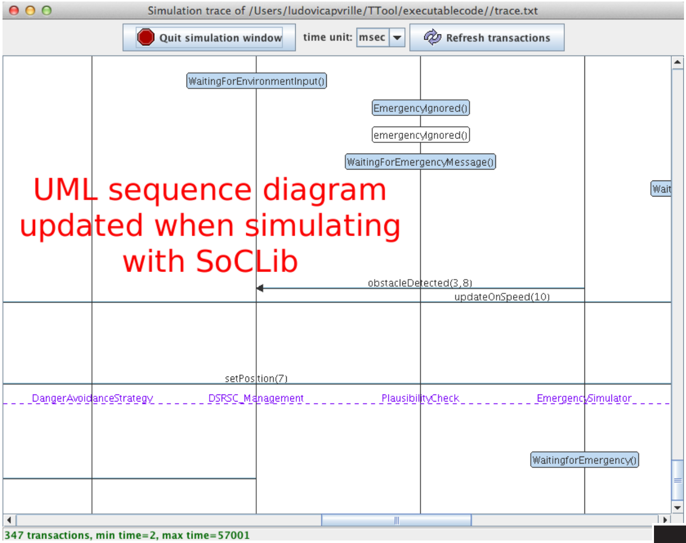
Virtual Prototyping: Console



```
 -> Locking mutex
DT> Adding pending request in inWaitqueue
DT - #629 time=75.998696832 block=NeighbourhoodTableManagement type=state_enteri
ng 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
```

(Virtual) Prototyping: Trace

TTool displays execution traces in a sequence diagram

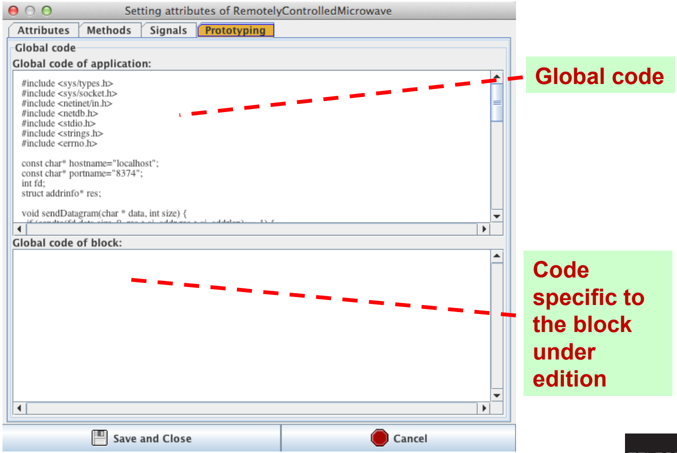


347 transactions, min time=2, max time=57001

UML sequence diagram updated when simulating with SoCLib

Customizing Generated Code with Your Own Code: Application and Block Code

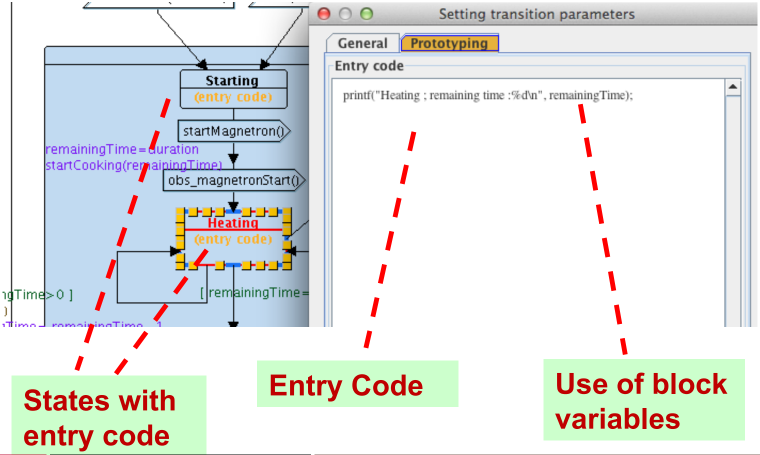
- Global code of the application
  - Inclusion of header files, global variables, ...
- Code global to one given block


Setting attributes of RemotelyControlledMicrowave
Attributes
Methods
Signals
Prototyping
Global code
Global code of application:
Global code of block:
Save and Close
Cancel
Global code
Code specific to the block under edition

```
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>
#include <stdio.h>
#include <strings.h>
#include <errno.h>

const char* hostname="localhost";
const char* portname="8374";
int fd;
struct addrinfo* res;

void sendDatagram(char * data, int size) {
```

Customizing Generated Code with Your Own Code: State Entry Code

- Code executed whenever a state is reached



```
printf("Heating ; remaining time :%d\n", remainingTime);
```

Use of Customized Generated Code

Console debug

- Using e.g. *printf()* function

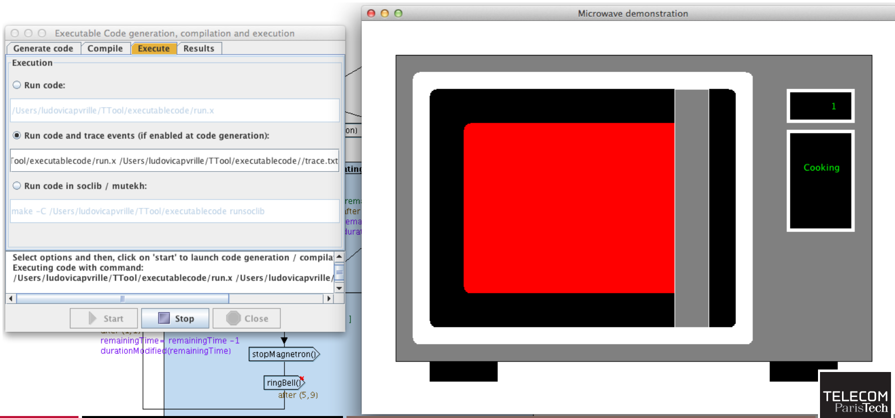
Connection to a graphical interface

- Piloting the code with a graphical interface
- Visualizing what's happening in the executed code
- Connection to graphical interface via, e.g., *sockets*

Use of Customized Generated Code (Cont)

Graphical interface for the microwave oven

- Socket connection to a graphical interface programmed in Java


Executable Code generation, compilation and execution

Generate code | Compile | Execute | Results

Execution

( ) Run code:
/Users/ludovicapvrille/TTool/executablecode/run.x

(●) Run code and trace events (if enabled at code generation):
Tool/executablecode/run.x /Users/ludovicapvrille/TTool/executablecode//trace.txt

( ) Run code in soclib / mutekh:
make -C /Users/ludovicapvrille/TTool/executablecode runsoclib

```
Select options and then, click on 'start' to launch code generation / compila
Executing code with command:
/Users/ludovicapvrille/TTool/executablecode/run.x /Users/ludovicapvrille/
```

To Go Further ...

TTool, DIPLODOCUS, AVATAR

ttool.telecom-paristech.fr