Institut
Mines-Telecom

Paris Sorbonne
University

Virtual Yet Precise Prototyping:

An Automotive Case Study

Daniela Genius, Ludovic Apvrille
daniela.genius@lip6.fr
ludovic.apvrille@telecom-paristech.fr

ERTSS'2016

Context

Model-oriented Design of Complex Embedded Systems

- Nowadays current practice in **software development** for embedded systems
- Hardware aspects less frequently designed using this kind of approach
- Models of software components are generally tested/executed on the local host, then integrated on target once available
- Even if several virtual prototyping platforms are available before the target is available, they require to re-model software elements in a different input format from the one used in models or programming languages

Virtual Prototyping

Related Work

- MARTE [Vidal et al. DATE 2009]
- AADL [IFIP-WADL 2004, Ben Youssef et al. ETSS 2014]
- Arcadia/Capella [Polarsys 2008, https://www.polarsys.org/capella/arcadia.html]

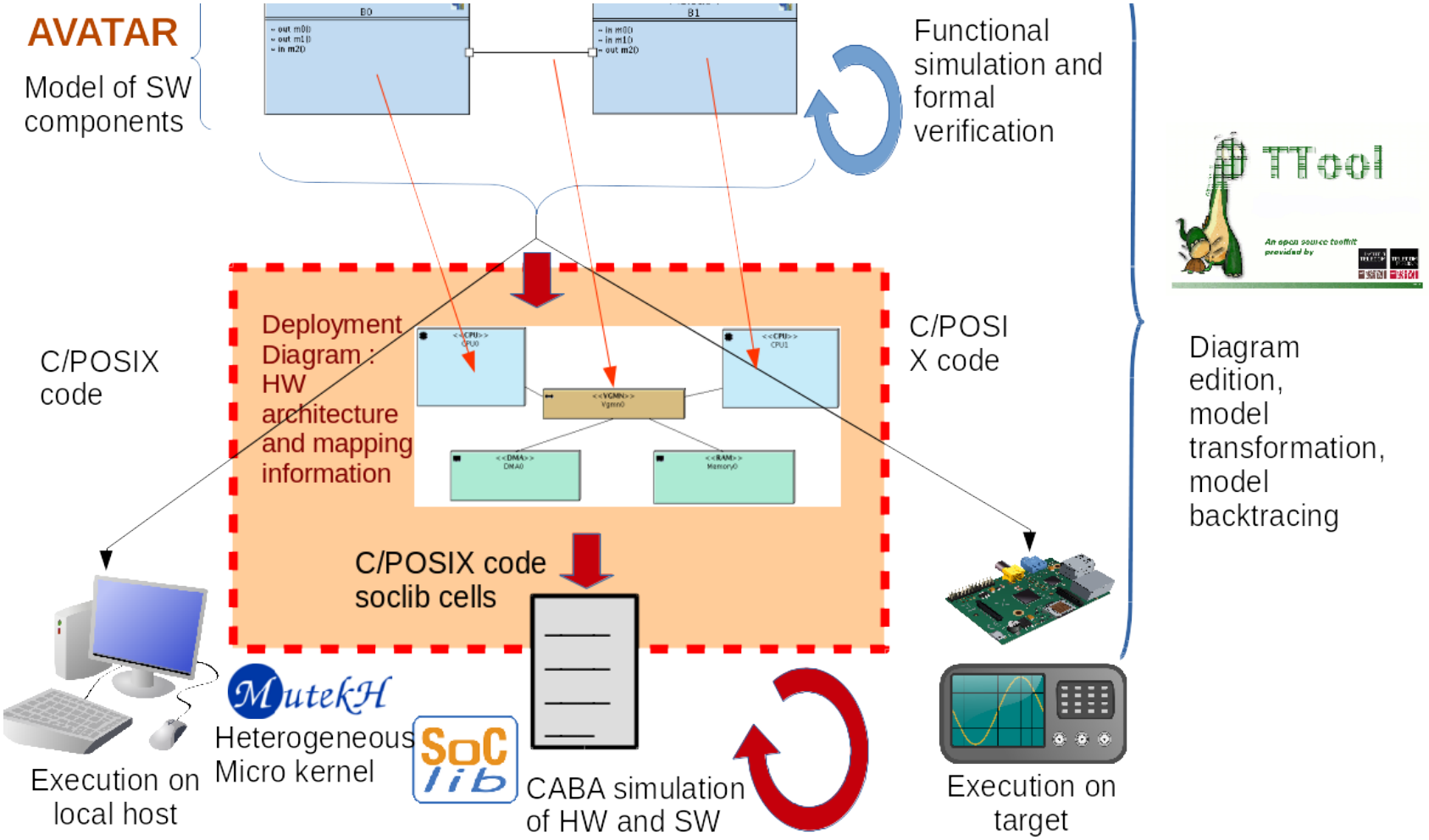
[ERTSS 2012]

- Aims at fully integrated way to model critical software components and candidate hardware architectures
- Joint use of both AVATAR and SocLib was not well integrated: description of the hardware execution environment only in textual form outside of SysML models

Overview of the Contribution: Method

1. Model refinement
2. Selection of OS, setting of options (scheduling algorithm, . . .)
3. Selection of a hardware platform and of 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: sequence diagram or debugger (e.g. gdb)

Overview of the Contribution: Tooling



AVATAR

- Model-oriented solution for the analysis and design of embedded software
- SysML diagrams describe the software aspects of the system, its safety and security properties
- Fully supported by the free software TTool
  - Edit AVATAR models
  - Simulation
  - Formal verification
- Like in most UML approaches, simulation and formal verification rely on a purely timed functional model, without considering any hardware target (CPU, bus, etc.)

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 thread support
- Note: any Operating System supporting POSIX threading that can be compiled for SoCLib could be used

Case Study: An Automotive Application

Automotive embedded system

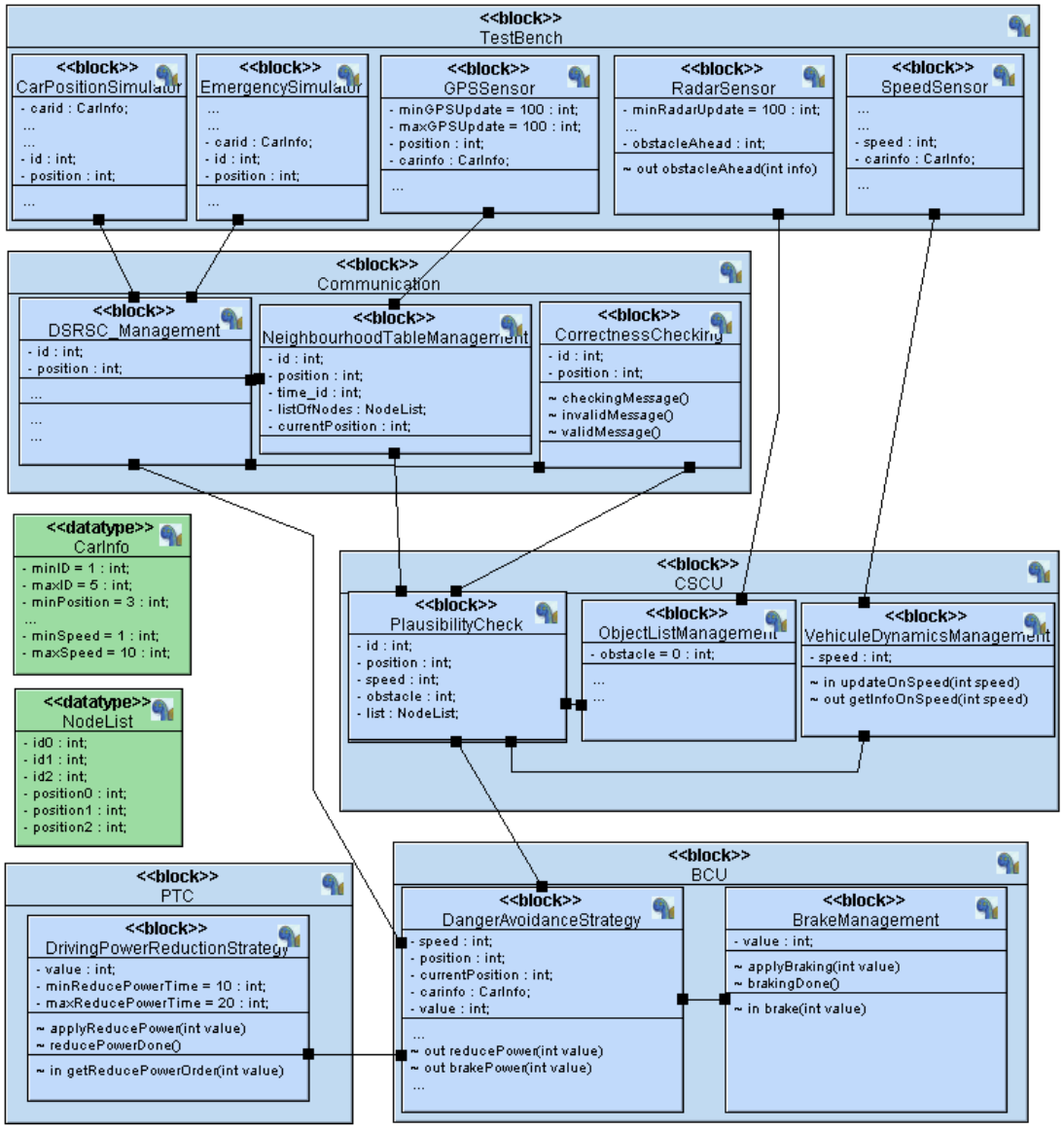
- Made upon a hundred of Electronic Control Units (ECUs)
- Interconnection with CAN / FlexRay / MOST

Automatic Braking Application

- Taken from Intelligent Transport System applications and from the EVITA european project

1. Obstacle is detected by a car
2. 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, ... $\rightarrow$ Plausibility check

Active Braking Use Case

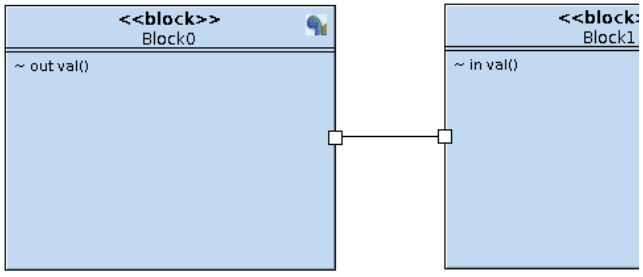


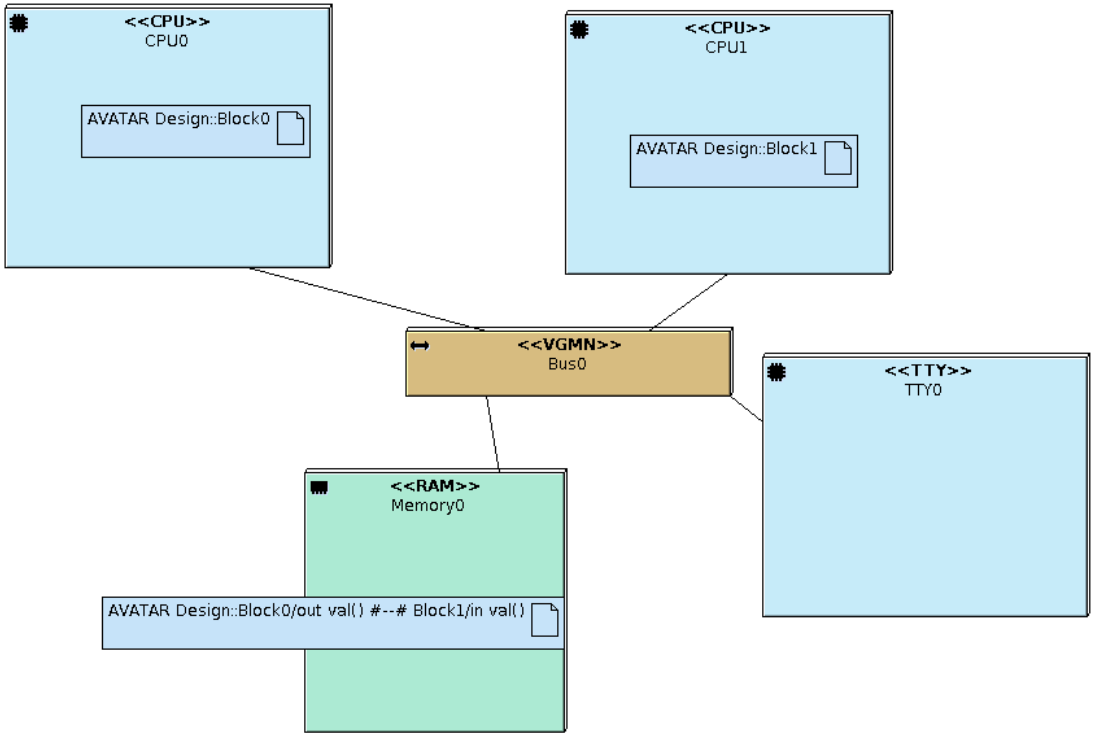
Deployment Diagram

- SysML representation of hardware components, their interconnection, tasks and channels
- Platforms can be modified in a matter of minutes
- A valid platform must contain at least one CPU, one memory bank and one terminal for observing the progress
- By choice, and to be discussed, some details (interrupt controller, simulation aides) are currently transparent to the user

Deployment Diagrams

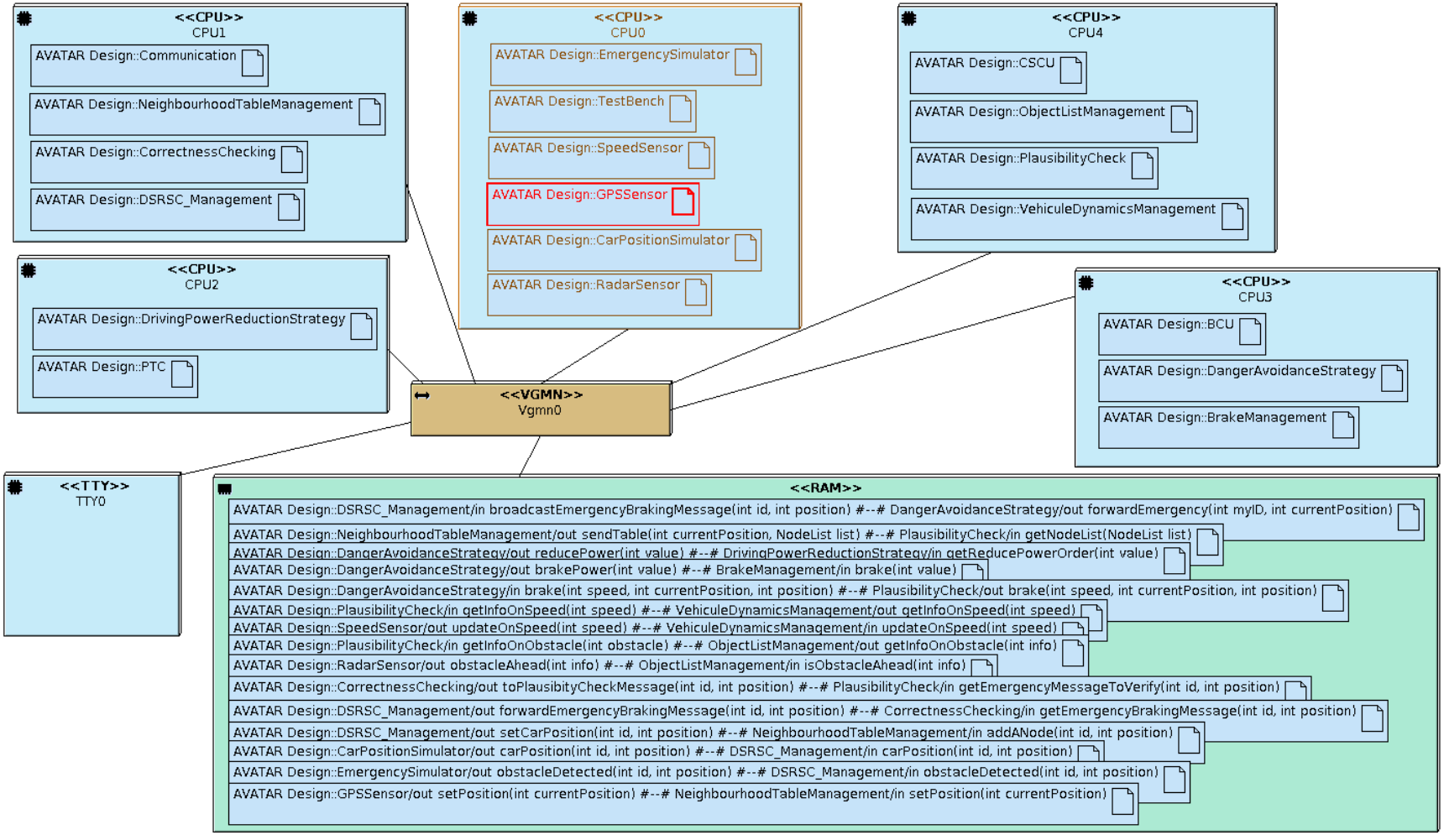
AVATAR specification





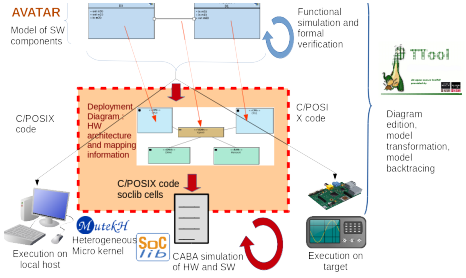
Deployment Diagram

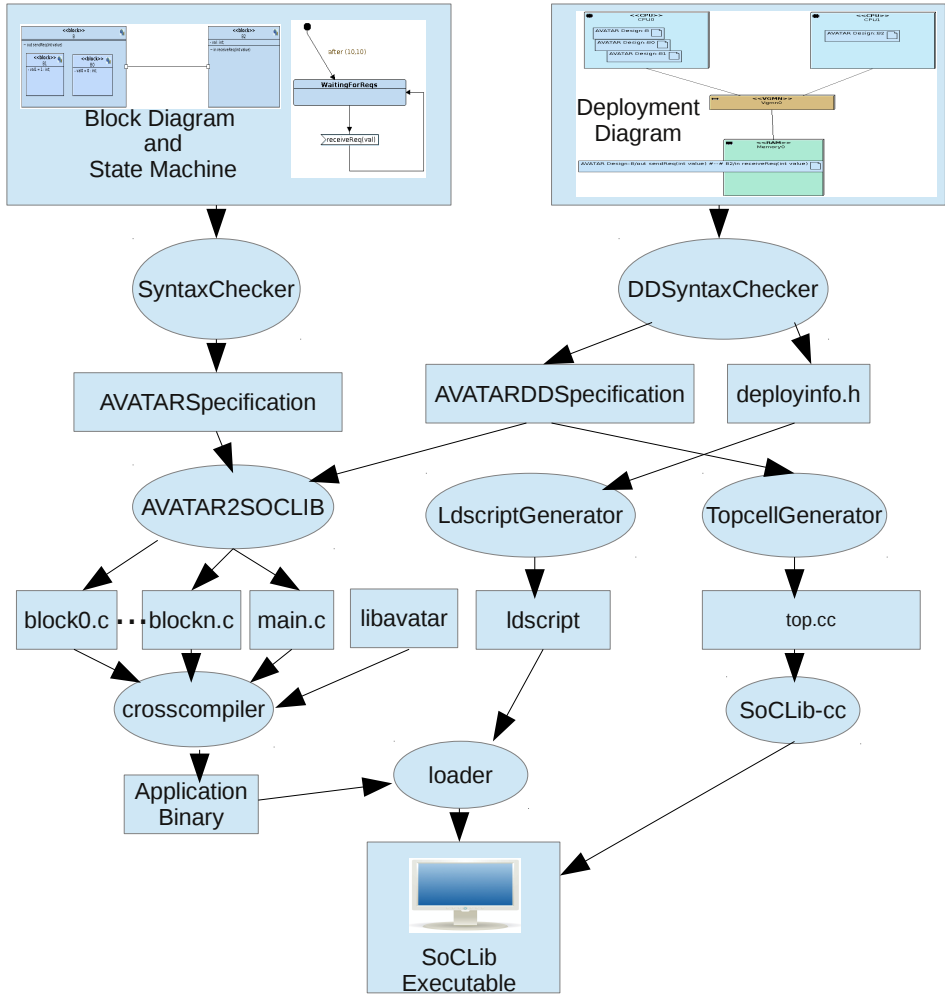
Deployment Diagram for the Case Study



New Tool Chain

Generate fully executable SoCLib specification for MPSoC from Deployment Diagram and software component diagrams





Model Transformation

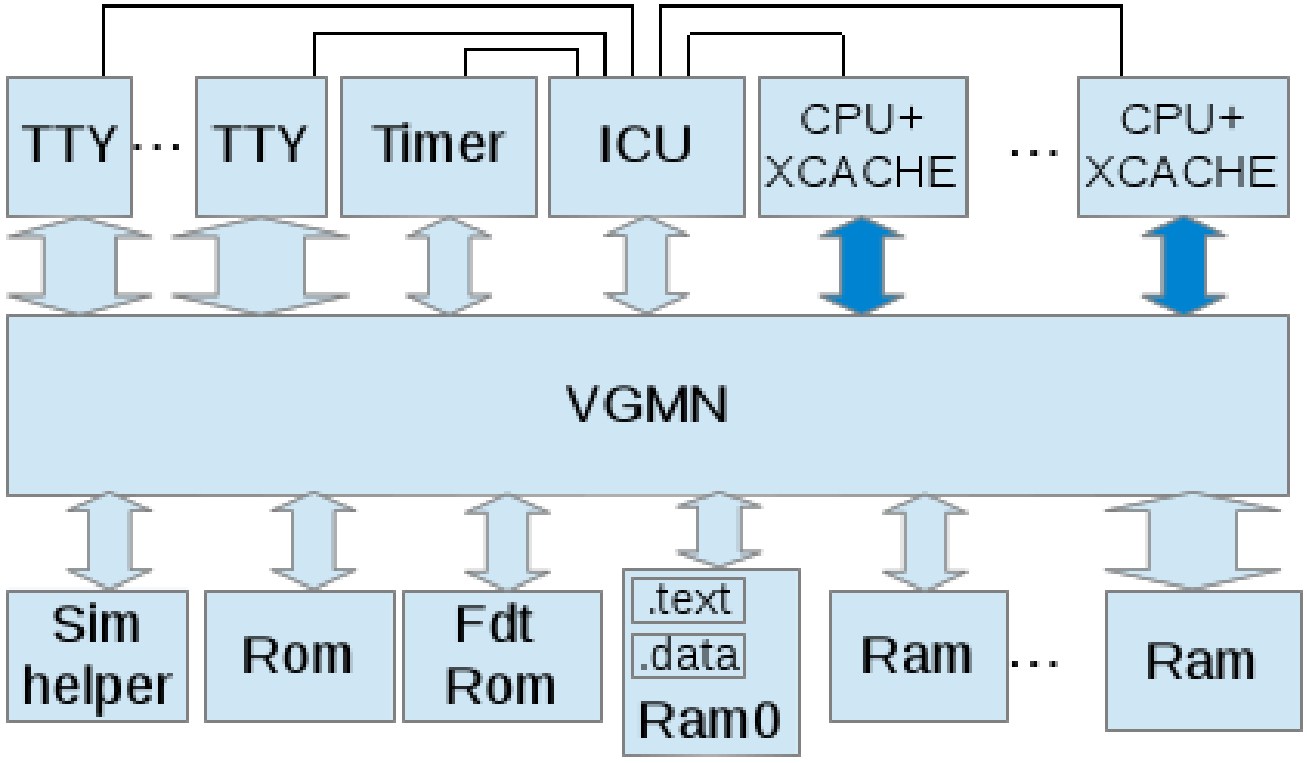
1. **Libavatar** Runtime for SoCLib, implements the AVATAR operators
2. **DDSyntaxChecker** checks the syntax of the deployment diagrams and identifies their elements
3. **AVATAR2SOCLIB** translates AVATAR blocks (i.e., software components) into C POSIX tasks and generates the main program
4. **TopcellGenerator** generates a SystemC top cell for cycle accurate bit accurate simulation
5. **LdscriptGenerator** generates the linker script, taking into account the mapping specified in the deployment diagram

The AVATAR Runtime

- Library of functions which capture the semantics of the AVATAR operators that appear in the code of the tasks (delay, asyncRead, etc.) and implements them using C/POSIX primitives
- Focuses on channels; interconnect latencies and cache effects make them difficult to implement efficiently on a MPSoC platform
- SoClib provides an efficient implementation of asynchronous channels as sowtware objects stored in on-chip memory, based on the Kahn model
- In order to run on a SoCLib platform, instead of the local workstation, the runtime had to be adapted

Prototyping: Top Cell Generation

- SystemC instanciation of components, netlist, table associating memory segments, charging of code
- Some components are transparent to the user (interrupts, simulation infrastructure)



Linker Script Generation

- Handling the mapping of channels is rather difficult since it is related to the main program, the top cell and the linker script (ldscript)
  - File which defines the memory layout;
  - Associates each entry section to an output section, which receives its address in memory
- Mapping information from Deployment Diagram is used to generate a file to be included by the ldscript generator

```
data_ldr.load_file(std::string(kernel_p)
 + ";.data;.channel0; ...
     .channel14;.cpudata;.contextdata");
```

Code Generation

Difficulties

- Implementing AVATAR channels with existing paradigm for SoCLib (software channel mapped to shared memory)
- Co-existence of synchronous and asynchronous channels
- Central request management required for synchronous channels

Extract from generated code for the DrivingPowerReductionStrategy block (1)

```
#include "DrivingPowerReductionStrategy.h"
static uint32_t _getReducePowerOrder;
...
void *mainFunc__DrivingPowerReductionStrategy(truct mwmr_s *channels_DrivingPowerReduc
tionStrategy[]){

 struct mwmr_s *DangerAvoidanceStrategy_reducePower__DrivingPowerReductionStrategy_get
ReducePowerOrder= channels_DrivingPowerReductionStrategy[0];

  int value = 0;
  int minReducePowerTime = 10;
  int maxReducePowerTime = 20;

  int __currentState = STATE__START__STATE;
  ...
  pthread_cond_init(&__myCond, NULL);
  fillListOfRequests(&__list, __myname, &__myCond, &__mainMutex);
```

Extract from generated code for the DrivingPowerReductionStrategy block (2)

```
  while(__currentState != STATE__STOP__STATE) {
    switch(__currentState) {
      case STATE__START__STATE:
      __currentState = STATE__WaitForReducePowerOrder;
      break;

      case STATE__WaitForReducePowerOrder:
      __params0[0] = &value;
      makeNewRequest(&__req0, 853, RECEIVE_SYNC_REQUEST,0,0,0,1, __params0);
      __req0.syncChannel = &__DangerAvoidanceStrategy_reducePower\
__DrivingPowerReductionStrategy_getReducePowerOrder;
      __returnRequest = executeOneRequest(&__list, &__req0);
      clearListOfRequests(&__list);
      DrivingPowerReductionStrategy__applyReducePower(value);
      __currentState = STATE__WaitForReducePowerToBePerformed;
      break;

      case STATE__WaitForReducePowerToBePerformed:
      waitFor(minReducePowerTime, maxReducePowerTime);
      DrivingPowerReductionStrategy__reducePowerDone();
      __currentState = STATE__WaitForReducePowerOrder;
      break;
    }
  }
  return NULL;
}
```

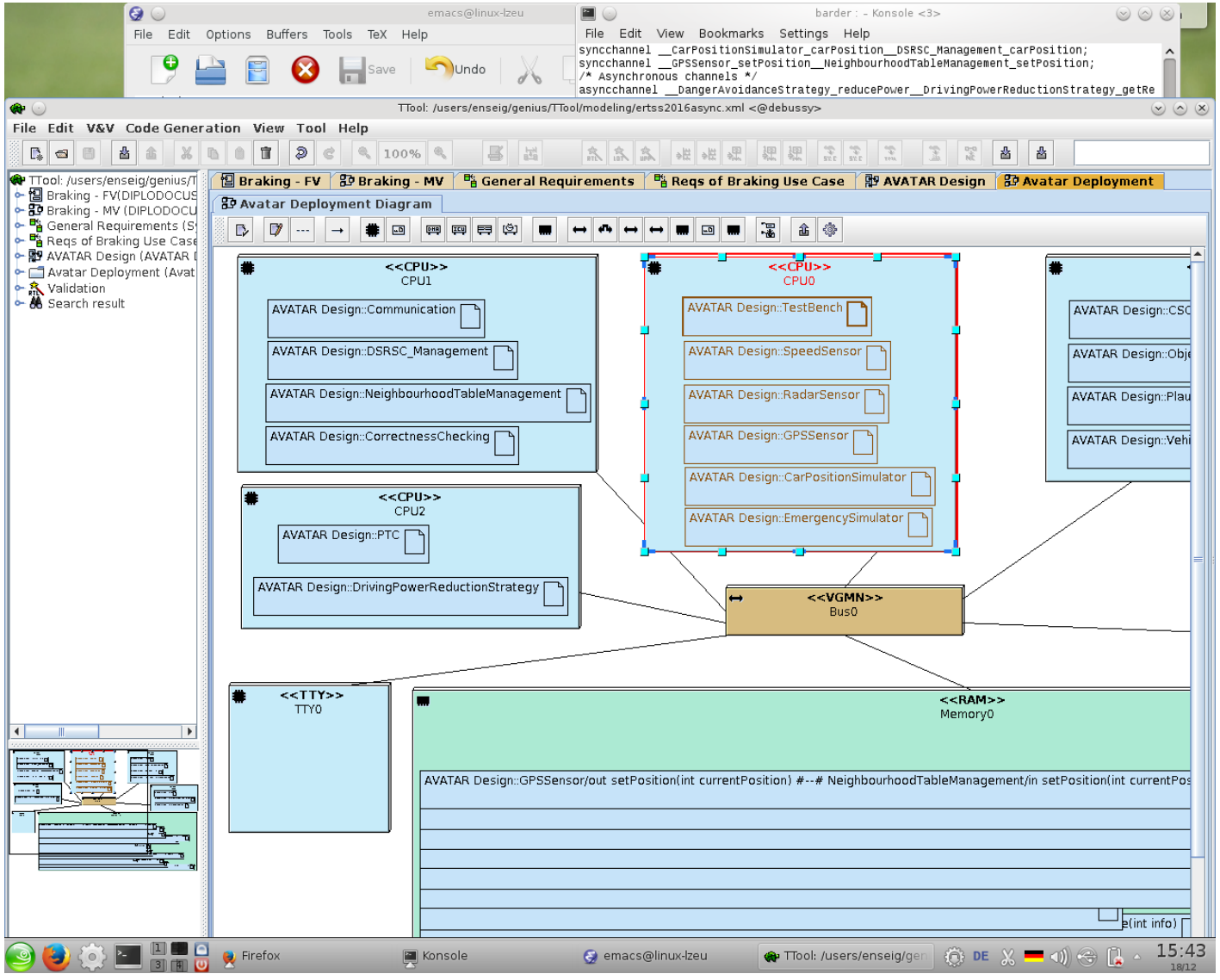
Code Generation: Declaration of Channels

```
syncchannel __DangerAvoidanceStrategy_reducePower
  __DrivingPowerReductionStrategy_getReducePowerOrder;

uint32_t *const DangerAvoidanceStrategy_reducePower
  __DrivingPowerReductionStrategy_getReducePowerOrder_lock= LOCKSADDR+0x0;

struct mwmr_status_s DangerAvoidanceStrategy_reducePower
  __DrivingPowerReductionStrategy_getReducePowerOrder_status
   =  MWMR_STATUS_INITIALIZER(1, 1);

uint8_t DangerAvoidanceStrategy_reducePower
  __DrivingPowerReductionStrategy_getReducePowerOrder_data[32*2];

struct mwmr_s DangerAvoidanceStrategy_reducePower
  __DrivingPowerReductionStrategy_getReducePowerOrder
  = MWMR_INITIALIZER(1, 1,
     DangerAvoidanceStrategy_reducePower__DrivingPowerReductionStrategy
    _getReducePowerOrder_data,&DangerAvoidanceStrategy_reducePower
    __DrivingPowerReductionStrategy_getReducePowerOrder_status,
  "DangerAvoidanceStrategy_reducePower
     __DrivingPowerReductionStrategy_getReducePowerOrder",
  DangerAvoidanceStrategy_reducePower__DrivingPowerReductionStrategy
  _getReducePowerOrder_lock);
```

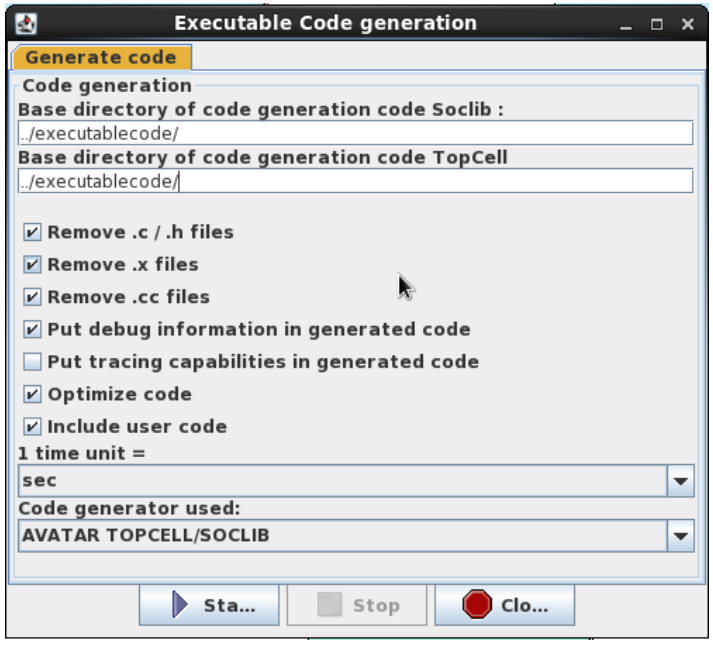
Code Generation: Extract from Main Program

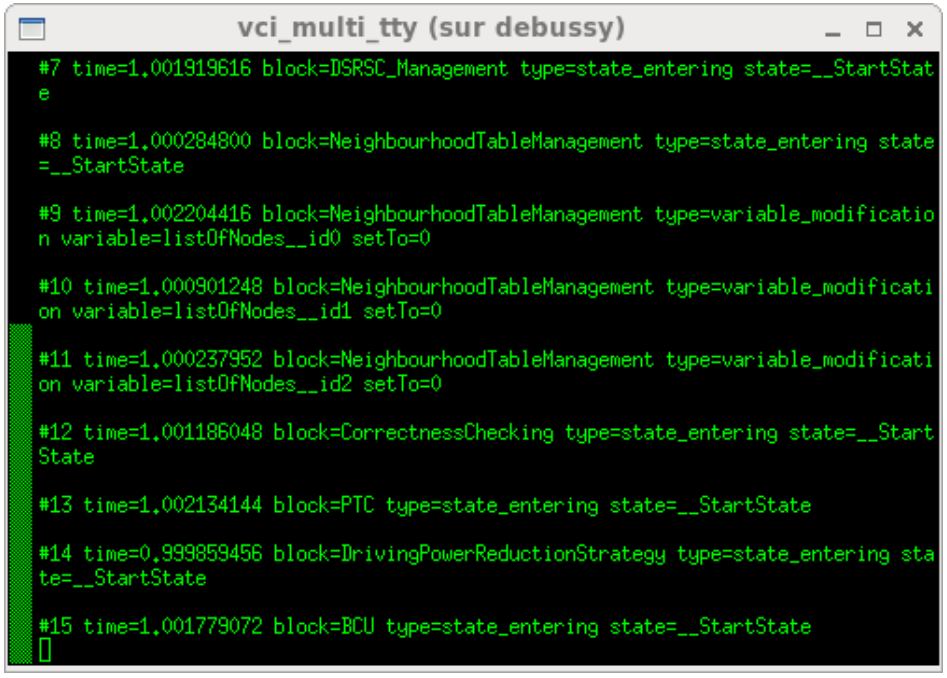
```
int main(int argc, char *argv[]) {
...
struct mwmr_s *channels_array_DrivingPowerReductionStrategy[1];
  channels_array_DrivingPowerReductionStrategy[0]
  =&DangerAvoidanceStrategy_reducePower__DrivingPowerReductionStrategy
       _getReducePowerOrder;

  ptr =malloc(sizeof(pthread_t));
  thread__DrivingPowerReductionStrategy= (pthread_t)ptr;
  attr_t = malloc(sizeof(pthread_attr_t));
  attr_t->cpucount = 2;

  debugMsg("Starting tasks");
  pthread_create(&thread__DrivingPowerReductionStrategy, NULL,
     mainFunc__DrivingPowerReductionStrategy,
     (void *)channels_array_DrivingPowerReductionStrategy);
```

Using TTool/SoCLib



Using TTool/SoCLib





Limitations

Currently Overcome

- Variations of the interconnect (bus, CAN, etc.)
- Central manager handling synchronous and asynchronous channels

Mid and longer term

- Simulation speed
- Currently PowerPCs only: crosscompilers
- Clustered interconnect

Future Work

Extensions

- Integration of application specific co-processors
- Animation and performance evaluation with Lip6 tools
- Feedback of results for Design Space Exploration

Long term : Design Space Exploration

- METROPOLIS
- SPADE
- SESAME
- ARTEMIS