Model-Based X, Virtual Prototyping, 42, SoCs, Sensor Networks, and Other Stories

Florence Maraninchi
www-verimag.imag.fr/~maranin
joint work with... the whole “synchronous” group

Verimag / University of Grenoble

14th SYNCHRON Open Workshop on Synchronous Programming 
Bamberg, november 26-30, 2007
Other Related Talks from the VERIMAG Synchronous Group

- Nicolas Halbwachs and Erwan Jahier: virtual prototyping in AADL and Lustre
- Tayeb Bouhadiba: the model of components 42
- Olivier Bezet: abstraction problems in virtual prototypes of sensor networks
- ...
1. What you can (already) do with the synchronous language Lustre (Summary)

2. Model-Based Engineering and Virtual Prototyping

3. 42, a Model of Components

4. Towards MBE and VP à la 42

5. Provocative Comments
Around Lustre...

Lustre

compilation

C Code
1 task, no OS
Around Lustre...

- Theorem Provers
- Abstract Interp.
- Simulation
- Interp.
- Model-Checker
- Compilation
- The system
  - its environment
  - Its safety prop.
- C Code
  - 1 task, no OS
- 1 task, no OS
- C Code

Lustre
Around Lustre...

- Automatic Test
- Theorem Provers
- Simulation
- Compilation
- Interp.
- C Code
- 1 task, no OS
- Model-Checker
- Abstract Interp.
- Interp.
- Safety prop.
Around Lustre...

Simulink/Stateflow

Automatic Test

Lustre
The system
its environment
its safety prop.

Compilation

Interp.

Simulation

C Code
1 task, no OS

Model-Checker
Abstract Interp.
Theorem Provers
Around Lustre...

- Simulink/Stateflow
- SystemC
- Automatic Test
- Model-Checker
- Abstract Interp.
- Theorem Provers

Lustre
- The system
- its environment
- Its safety prop.

Compilation

Interp.

Simulation

C Code
- 1 task, no OS

Maraninchi (Verimag, Grenoble)
Around Lustre...

- Simulink/Stateflow
- SystemC
- Automatic Test
- Model-Checker Abstract Interp. Theorem Provers
- Theorem Provers
- Abstract Interp.
- Simulation
- Multi-task code for RTOS or TTA
- C Code 1 task, no OS
- Compilation
- Compilation
- Interp.
- Interp.
- Lustre The system its environment Its safety prop.
- 1 task, no OS
- Maraninchi (Verimag, Grenoble)
Around Lustre...

- Simulink/Stateflow
- SystemC
- Sensor networks, middleware (avionics, space)
- Modeling
-自动测试
-自动测试
-模型检查器
-抽象解释器
-模拟
-编译

Lustre

The system
its environment
Its safety prop.

C Code
1 task, no OS

Multi-task code for RTOS or TTA
Around Lustre...

- Simulink/Stateflow
- Model-driven
  fully automatic
development
  (for a specific exec platform)
- Multi-task code
  for RTOS or TTA

Lustre
Around Lustre...

- Simulink/Stateflow
- SystemC
- Compilation
- Lustre or SMV
  - The system
  - Its environment
  - Its safety prop.
- Model-Checker
  - Abstract Interp.
  - Theorem Provers
- Lustre as an executable
  - semantics definition
  - formalism
Around Lustre...

Lustre or ReactiveML

Sensor networks, middleware (avionics, space)

Modeling

Interp.

Simulation

Automatic Test

semantics–inside Virtual Prototyping

Maraninchi (Verimag, Grenoble)

MBX, VP, ...

Synchron 07 4 / 45
Current Research Directions and Industrial Collaborations

- Language Design, Code Generation (Lustre, AOP, ...)
- Formal Verification (Model-checking, abstract interpretation)
- Automatic Test-Case Generation from a Model of the Environment
- Model-Driven and Virtual Prototyping with synchronous languages and/or engineering languages (SystemC, Simulink, ...)
- Component-Based Approaches

Past or Ongoing Projects with:
- EsterelTechnologies, Polyspace, Airbus, Schneider Electric, STMicroelectronics, Thomson, Astrium-EADS, Audi, Renault, RATP, FranceTelecom R&D, Coronis Systems, KeesDA, TTTech, DOCEA Power, ...
+ Minalogic Partners
What you can (already) do with the synchronous language Lustre (Summary)

General (Simplified) Picture
- Virtual Prototyping with Transactional Models
- Virtual Prototyping of Sensor Networks with Synchronous Languages and Tools
- On Abstraction and Faithfulness

42, a Model of Components

Towards MBE and VP à la 42

Provocative Comments
Model-Based Engineering and Virtual Prototyping

- General (Simplified) Picture
- Virtual Prototyping with Transactional Models
- Virtual Prototyping of Sensor Networks with Synchronous Languages and Tools
- On Abstraction and Faithfulness
Model-Based Eng. vs Virtual Prototyping

High-Level Models
Model-Based Eng. vs Virtual Prototyping

High-Level Models

Implementation
Model-Based Eng. vs Virtual Prototyping

High-Level Models

(automatic) transformations

Comparisons

Implementation

Maraninchi (Verimag, Grenoble)
Model-Based Eng. vs Virtual Prototyping

High-Level Models

(automatic) transformations

Comparisons

Implementation

An Example
(Caspi et al., VERIMAG)

Simulink
(plant+controller)

V&V

Lustre

Compilation

Code for a RTOS, or for a TTA
Model-Based Eng. vs Virtual Prototyping

High-Level Models
Model-Based Eng. vs Virtual Prototyping

Executable High-Level Models (+ non–func aspects) → Simulator

results
(implement. parameters, performance estimations, ...)
Example 1:

**Transaction-Level Modeling (TLM) of Systems-on-a-Chip (with SystemC)**
Model-Based Eng. vs Virtual Prototyping

Example 2:
Virtual Prototyping of Sensor Networks with NS, NAB, opnet, ...
(modeling energy consumption)
Model-Based Eng. plus Virtual Prototyping

Example 3 (FM et al., VERIMAG):
(semantics inside, component-based)
Virtual Prototyping of Sensor Networks
(fine-grain modeling of energy consumption)
Model-Based Eng. plus Virtual Prototyping

Executable
High-Level Models
(+ non–func aspects)

(automatic) transformations
Comparisons
Implementation

Simulator
results
(implem. parameters, performance estimations, ...)

Not necessarily the ultimate goal

Marinichchi (Verimag, Grenoble)
2 Model-Based Engineering and Virtual Prototyping

- General (Simplified) Picture
- Virtual Prototyping with Transactional Models
- Virtual Prototyping of Sensor Networks with Synchronous Languages and Tools
- On Abstraction and Faithfulness
The MINALOGIC/openTLM Context

A Minalogic project (90 person.year)
open-source tools for transaction-level modeling of systems-on-a-chip.
STMicroelectronics, Thomson, KeesDA, Orange-Silicomp,
CEA-LETI, TIMA, INRIA, VERIMAG

STMicroelectronics + VERIMAG working group since 2002:
3 PhDs, 1 starting 01/08
some publications: EMSOFT’05, ACSD’05, J. DAES 06, FMCAD’06,
FMICS’06, SPIN’07, DATE’08
The openTLM Context: Systems-on-a-Chip

- RTL
- Synthesizable+
- Cycle and Data Accurate+
- Slow Simulations−

Maraninchi (Verimag, Grenoble)
The openTLM Context: Systems-on-a-Chip

TLM
+ Early Available
+ Fast Simulations
– Not synthesizable

Bit Accurate, but:
asynchronous,
non-deterministic
No automatic
transformations

RTL
Synthesizable+
Cycle and Data Accurate+
Slow Simulations−
Virtual Prototyping for other Systems

TLM
+ Early Available
+ Fast Simulations
Virtual Prototyping for other Systems

A Virtual Prototype of the Embedded Execution Platform

TLM
+ Early Available
+ Fast Simulations

Main problems:
-- Faithfulness of the model
-- Automatic Testing of the Embedded Software
2 Model-Based Engineering and Virtual Prototyping

- General (Simplified) Picture
- Virtual Prototyping with Transactional Models
- Virtual Prototyping of Sensor Networks with Synchronous Languages and Tools
- On Abstraction and Faithfulness
Sensor Networks and Energy Consumption

Several thousands of sensors communicating by radio + a special node connected to the network (the sink).

Examples: detection of a radioactive cloud, ...
The node itself

- a radio
- a sensor
- a CPU
- a memory
- a battery
The Structure of the Virtual Prototype

Physical Environment

One node

Sensor

Appli

Routing

Power Manage.

MAC

Radio

Channel (Topo)

Perturb.

Rand.

Marinich (Verimag, Grenoble)
The Structure of the Virtual Prototype

Physical Environment

Sensor
Appli
Routing
Power Manag.
RTC
MAC
Radio
Sensor
CPU
Mem.
MAC
Radio
RTC
E. Sum
Battery (init.E)
Charge mode
Physical Environment

Radio
Channel (Topo)
Perturb.
Rand.

Environment charging (night/day...)

Maraninchi (Verimag, Grenoble)
The Structure of the Virtual Prototype

One node

- Sensor
- Appli
- Routing
- Power Manag.
- RTC
- MAC
- Radio
- Sensor
- CPU
- Mem.
- MAC
- Radio
- RTC
- E. Sum
- Battery (init. E)
- Charge mode
- Physical Environment
- Environment charging (night/day...)
- Physical Environment
- Channel (Topo)
- Perturb.
- Rand.
- A discrete Energy model

Marinichi (Verimag, Grenoble)
The Structure of the Virtual Prototype

![Diagram](image-url)
The Structure of the Virtual Prototype

A discretized non-deterministic model of the physical environment

One node

Sensor
Appli
Routing
RTC
MAC
Radio

Power Manag.

Sensor
CPU
Mem.
MAC
Radio
RTC

Physical Environment

Charge mode

Environment
(charging)
(night/ day...)

Power Manag.

Channel
(Topo)

Perturb.

A discretized non-deterministic model of the physical environment
All these models are expressed in the same MoCC (synchronous parallelism)
Comments (1)

This virtual prototype (written in ReactiveML + Lurette, thanks to L. Mandel, P. Raymond, E. Jahier) is quite precise:

- It contains the "real" code (a ReactiveML re-implementation of the C protocols)
- The energy models are taken from the documentation of the providers (the radio, the CPU, ...)
- The model of the physical environment is given in some operational way (a generator of stimuli)
- The model of the radio channel is taken from the literature on networks (a precise one, with fading and shadowing, not the simple "disk" model)
- We simulate the asynchrony between the nodes
Energy consumption is associated with "states" in the energy models of all the hardware pieces $\neq$ energy associated with abstract events like "a message reaches the sink"

Stimuli on the sensors are spatially and temporally correlated, hence more realistic than independent Poisson Laws on the sensors

The complete model is too complex to be analysed in full details, but some abstractions can be derived from this very precise model.
Model-Based Engineering and Virtual Prototyping

- General (Simplified) Picture
- Virtual Prototyping with Transactional Models
- Virtual Prototyping of Sensor Networks with Synchronous Languages and Tools
- On Abstraction and Faithfulness
Models are Necessarily Abstract (see J.-L. Borges)

...In that Empire, the craft of Cartography attained such Perfection that the Map of a Single province covered the space of an entire City, and the Map of the Empire itself an entire Province. In the course of Time, these Extensive maps were found somehow wanting, and so the College of Cartographers evolved a Map of the Empire that was of the same Scale as the Empire and that coincided with it point for point. Less attentive to the Study of Cartography, succeeding Generations came to judge a map of such Magnitude cumbersome, and, not without Irreverence, they abandoned it to the Rigours of sun and Rain. In the western Deserts, tattered Fragments of the Map are still to be found, Sheltering an occasional Beast or beggar; in the whole Nation, no other relic is left of the Discipline of Geography.
Abstraction vs Faithfulness

(see also: talk by O. Bezet)

A Faithful model...
for hiking or riding
but not so good for driving

The real system
Abstraction vs Faithfulness

- Abstract Model
- Faithfulness
- Real System
Abstraction vs Faithfulness

Abstract Model

Comparisons are possible

Very precise (operational) model

Faithfulness Problem

Real System

Faithfulness problem
Modular Worst-Case Energy Consumed (SLAP’07)

- The complete model of a sensor network is far too complex to be analysed as a whole. Hence we need abstractions, and they should be modular.

- A notion of precision in energy models, captured by an order: \( M_1 \leq M_2 \). The evaluation of this order for two given energy models should be made automatic (with approximate verification methods).

- A pre-congruence property, for all composition operators \( \text{op} \) used in the model: \( M_1 \leq M_2 \iff \forall M. \text{op}(M, M_1) \leq \text{op}(M, M_2) \). This property has to be proved by hand, once and for all. Highly depends on the separation between function and energy.
The Lego Principle

Globally: A model $M$
The Lego Principle

...
The Lego Principle

Globally: A model $M'$ more precise then $M$
1. What you can (already) do with the synchronous language Lustre (Summary)

2. Model-Based Engineering and Virtual Prototyping

3. 42, a Model of Components
   - Context, Motivations and Related Work
   - 42 in a Nutshell

4. Towards MBE and VP à la 42

5. Provocative Comments
42, a Model of Components

- Context, Motivations and Related Work
- 42 in a Nutshell
42, Model-Based X, Virtual Prototyping...

(see also: talk by T. Bouhadiba)

42 is a tool for reasoning on what MBX and VP mean for embedded systems, looking at several applications domains and domain-specific methods:

- Embedded Control (Simulink, synchronous formalisms)
- Systems-on-a-chip (Transaction-Level Modeling, SystemC)
- Sensor Networks (VP with NS, opnet, dedicated simulators, lanugaes from the synchronous family)
42 aims at enforcing the FAMAPASAP Principle

The “Forget As Much As Possible As Soon As Possible” principle:

- Closing the box as soon as possible
- Identifying the component data and control ports that have to be exposed
- Giving a precise specification of the components with protocols
An Example Embedded System with Components
An Example Embedded System with Components

On a processor:

- A real-time OS + several processes
- Each process can be programmed in a component-based framework

Hardware IP’s (components)
Main Sources of Inspiration (1)

Components are **actors**, assembled with wires
- A **director** defines how they behave together.
  - A director implements a MoC.
- Hierarchic framework
- There are a lot of available MoCs

ptolemy.eecs.berkeley.edu
Main Sources of Inspiration (2)

Modeling heterogeneous systems with Synchronous Lgs

A Lustre program that models 2 processors, each of them running a scheduler and 2 threads.

“Virtual Execution of AADL Models via a Translation into Synchronous Programs”, EMSOFT’07

Lustre model: formal semantics + executability
(All MoCs are encoded into the synchronous one, using additional signals)
3 42, a Model of Components

- Context, Motivations and Related Work
- 42 in a Nutshell
42 in a Nutshell: a Basic Component

Input Data Ports:
- id1
- id2
- id3

Control Ports:
- ic1
- ic2

Output Data Ports:
- od1
- od2
- od3

Input Control Ports:
- oc1
- oc2

Output Control Ports:
- id1
- id2
- id3

Maraninchi (Verimag, Grenoble)
For ic1:

```java
a = id1.get();
b = id2.get();
od1.set(f(a, b));
oc1.set(false);
```

atomic in any context where the component is used.
42 in a Nutshell: Assembling Components
42 in a Nutshell: Assembling Components
42 in a Nutshell: Assembling Components

Diagram of components with input and output ports:
- Comp A:
  - Input port (a)
  - Output port (d)
- Comp B:
  - Input port (b)
  - Output port (c)
- Comp C:
  - Input port (e)
- Comp D:
  - Input port (f)

ic, oc

input port
output port

Maraninchi (Verimag, Grenoble)
42 in a Nutshell: Assembling Components

The controller:

- Comp A
- Comp B
- Comp C
- Comp D

Input ports: ic, oc
Output ports:

Maraninchi (Verimag, Grenoble)
MBX, VP, ...
Synchron 07
**42 in a Nutshell: Assembling Components**

The controller:
- dialogs with ABCD (ic, oc)

---

The diagram shows a network of components labeled A, B, C, and D, with input and output ports indicated by the symbols `ic, oc`. The components are connected by arrows indicating data flow and interaction, with labels such as `a`, `b`, `c`, `d`, and `e`.
42 in a Nutshell: Assembling Components

The controller:
- dialogues with ABCD (ic, oc)
- Manages memory for a, b, c, d, e, f
42 in a Nutshell: Assembling Components

The controller:

- defines glob. ic, oc, id, od
Controller is

```plaintext
var m : bool = true ;
for X do :
    m_a, m_b, m_c: FIFO(1,int);
    m_d, m_e, m_f: FIFO(4,int);
    if (m) {
        m_a.put ; // reads i1
        m_a.get ; D.z ; m_f.put ; m_f.get ;
        A.u; m_b.put; m_d.put;
        m_b.get; B.v; m = m or p ;
        m_c.put ; m_c.get ;//defines o1
        m_d.get ; C.w ; m_e.put ;
        C.y ; m_e.put ;
        m_e.get ; D.k ; m_e.get ;
        D.k ; m = ! m ;
    } else { ... }
    yy = true ;
```
Component Protocols and Checking Assemblages

related work: OO protocols, circuit sequential “don’t cares”, circular assume/guarantee rules, Signal clock+data specifications, ...
Component Protocols and Checking Assemblages

* Sequential constraints
  (final states define the macro-steps)

related work: OO protocols, circuit sequential “don’t cares”, circular assume/guarantee rules, Signal clock+data specifications, ...
**Component Protocols and Checking Assemblages**

- **Data dependencies**
  - \((id1 \text{ and } id2)\)
  - \((od1)\)

- **Sequential constraints**
  - Final states define the macro-steps

- **Data dependencies**

---

**related work:** OO protocols, circuit sequential “don’t cares”, circular assume/guarantee rules, Signal clock+data specifications, ...
Component Protocols and Checking Assemblages

* Sequential constraints
  (final states define the macro−steps)

* Data dependencies

* Control information
  (stored for further use in this automaton)

related work: OO protocols, circuit sequential “don’t cares”, circular
assume/guarantee rules, Signal clock+data specifications, ...
Component Protocols and Checking Assemblages

(id1 and id2)  
(op /x:=ctl)  
(od1)  

(id1 and IF x THEN id2)  
(op2)  
(IF not x THEN od2)  

* Sequential constraints  
(final states define the macro-steps)  

* Data dependencies  

* Control information  
(stored for further use in this automaton)  

* Conditional data expressions

related work: OO protocols, circuit sequential “don’t cares”, circular assume/guarantee rules, Signal clock+data specifications, ...
1. What you can (already) do with the synchronous language Lustre (Summary)

2. Model-Based Engineering and Virtual Prototyping

3. 42, a Model of Components

4. Towards MBE and VP à la 42

5. Provocative Comments
A Problem

A team looking at:
An avionics execution platform (buses, several CPUs, some of them running RTOS, sensors, actuators) and implementation problems (taking C code for individual tasks)

Others teams looking at:
SCADE or Lustre or SAO (CAS) descriptions of individual tasks, eventually compiled into C
A Problem

A team looking at:
An avionics execution platform
(buses, several CPUs, some of them running RTOS, sensors, actuators)
and implementation problems
(taking C code for individual tasks)

Others teams looking at:
SCADE or Lustre or SAO (CAS)
descriptions of individual tasks,
eventually compiled into C

Provider of SW components, to be used in some (implicit) intended context of use

the system-level view,
need for VP (at least) and glue generation
Current Practise

Towards MBE and VP à la 42

The HW platform (CPUs, bus, ...)

C code

SCADE, C, SAO, ...

Implementation

Manually done:
- mapping tasks–CPUs
- glue code for the OS

Maraninchi (Verimag, Grenoble)

MBX, VP, ...

Synchro 07 39 / 45
42 as the system-level description language

A 42 system−level description.

The HW platform (CPUs, bus, ...)

Towards MBE and VP à la 42
42 as the system-level description language

A 42 system-level description.

A 42 MoC expressing: a SW component runs on a HW component

The HW platform (CPUs, bus, ...)

TLM-like modeling task

42ized code

SCADE, C, SAO,...
42 as the system-level description language

A 42 system−level description.
42 as the system-level description language

A 42 system-level description.

Virtual Prototype Execution
Towards MBE and VP à la 42

42 as the system-level description language

A 42 system-level description.

Automatic glue-code generation

The HW platform (CPUs, bus, ...)

Maraninchi (Verimag, Grenoble)
To be Done...

- 42-ization of the TLM principles (mimicking SystemC behaviour, or not)
- Invent the 42 MoC that means: a **SW component** runs on a **HW component**
- Choose a realistic execution platform and model it
- Understand the manual implementation method (relies on *patterns* for a quite restrictive class of execution platforms)
- Mimick the manual method with some automatic generation method
- Apply this to a case-study (with a SystemC model of the execution platform because we don’t have the real HW!)
1. What you can (already) do with the synchronous language Lustre (Summary)

2. Model-Based Engineering and Virtual Prototyping

3. 42, a Model of Components

4. Towards MBE and VP à la 42

5. Provocative Comments
Provocative Comment (1)

Forget about non executable models

Choice is between:

Start from executable models, then abstract them (possibly turning them into non-deterministic models) and formalize their MoC

(Simulink, SystemC, opnet, synchronous languages used as modeling languages, ...)

vs

Define high-level non executable (and semantically loose) models and then try to “animate” them (defining the semantics as a side-effect)

(UML, AADL, SysML, ...)

Maraninchi (Verimag, Grenoble)
Provocative Comment (2)
Making everything automatic: no hope!

A model of the application SW +
A model of the execution platform (HW+OS)
⇒ magic ⇒
An implementation, including a mapping of the SW onto the HW
Provocative Comment (2)
Making everything automatic: no hope!

A model of the application SW +
A model of the execution platform (HW+OS)
⇒ magic ⇒

An implementation, including a mapping of the SW onto the HW

Instead: Design an abstract model of a class of execution platforms and use it for:
— VP together with the SW part
— Dedicated implementation generation algorithms
Provocative Comment (3)
The main problem is in MoCCs, not languages or
The language semantic problem is the easy part

We don’t really need more languages, but we need to understand the MoCCs behind the existing ones.
We need MoCCs for system-level-descriptions (HW, SW, OS, physical environments, ...