HW/SW Cyber-System
Co-Design and Modeling

Julio Oliveira

Karol Desnos
Who are we?

Julio de Oliveira

Position:
• TNO - Researcher & innovation scientist

Topic of interest:
• Large, distributed, and autonomous systems

Karol Desnos

Position:
• INSA Rennes – Assoc. Prof.
• IETR - Researcher

Topic of interest:
• Dataflow Programming
• Embedded MPSoCs
Text-book definitions for Cyber-Physical Systems:

• CPS are complex systems integrating:
  – Computation processes
  – Network of communication
  – Physical entities (actuators and sensors, time, mechanics, temperature, ..., and you!)

• CPS is an **engineering** discipline, focused on technology, with a **strong foundation in mathematical abstractions**.

source: Berkeley CPS website, http://cyberphysicalsystems.org/
Abstraction?

• Tradeoff between level of details and complexity.
What is a model?

- Abstract (mathematically grounded) representation capturing predictable characteristics of a “system”.
- Models for similar constituents may take many forms.
  - To capture different characteristics.
  - To be more suitable for a different system size.

Molecules

Ideal Gas Law

\[ pV = nRT \]

Meteorology

*: Physicist way of saying magic number
CPS Co-Design?

- “Old” embedded system co-design flow:

  - Where are physical concerns?

  - Where are physical concerns?

  - Time to market
CPS Co-Design?

• Model-based CPS co-design flow:

- Models enable: system assessment, system validation & verification, (automated) system implementation.

![Diagram showing the co-design flow with HWs, SWs, and Physical layers.](image)

- Requirements & specification
- Component Design
- Implementation

Model-based Verif. & Valid.
Model-based Verif. & Valid.

Time to market
CPS design is complex

• Many building blocks,
• Separate but intricate optimization objectives,
• Many design constraints.

• **Application Constraints**
  • Real-time requirements
  • Reliability constraints
  • Limited size and power

• **Cost Constraints**
  • Engineering cost
  • Production cost

• **External Constraints**
  • Regulation and Standards
  • Environmental constraints
Course objective

1. HW/SW Cyber-System Co-Design and Modeling
   Get a sense of how, why, and which models are used at different levels and steps of CPS design.

2. HW/SW Cyber-System Modeling Tools
   What do tools do to (automatically) build actual CPS systems from abstract models.
System Level Modeling
Modeling as an engineering activity

Abstraction
(Simplification)

Description
(Specification)

Operational
(Executable)
System-level Modeling

Modeling – Example

\[ \frac{dP}{dt} = \kappa P \left(1 - \frac{P}{K}\right) \]
Why modeling CPS (SoS) is challenging?

Which abstraction?
How to describe?
Operational?

Complexity!
What we mean by complexity?
Why bother?

System-level Modeling

Algorithm

High-level desc.

Architecture

High-level desc.

Multicore Compiler

Portable Multicore Program

Simulator + Debugger + Profiler

Multicore Runtime

Karol Desnos (IETR) & Julio Oliveira (TNO)
System-level Modeling

Why bother?

Source: EU DEMANES project
System-level Modeling

Why bother?

Source: INRIA
In a nutshell

System-level Modeling

Modeling requires:
- Abstraction
- Description
- Operational

CPS, SoS, Hw/Sw systems are complex due to:
- Multiple views
- Communication between stakeholders
- Analysis
- Decision making
- Implementation

Mission of Modeling Engineer:
- Obtain a modeling methodology for CPS / SoS

Karol Desnos (IETR) & Julio Oliveira (TNO)
Ways to approach system level modeling
Approach 1: Model for the task in hand

System-level Modeling

Karol Desnos (IETR) & Julio Oliveira (TNO)
Approach 1: Model for the task in hand

7. Procedure

This section describes the steps to be followed by the user in the Oracle Application with detail screen shots. After successful log in into the Oracle Application the user has to follow the following navigation to create a manual/standalone invoice in the system.

Prerequisite: Before navigating to the application the user should have following:
- Original copy of the vendor invoice.
- Copy of the manual PO/WO.
- Certificate of completion/ Proof of receipt of goods.
Model for the task in hand fails

Major problem for the development productivity

Introduction of errors:
Human failure or mis-interpretation

Almost impossible to optimize at system level
Approach 2: Model transformation

A model transformation is an automated way of modifying and creating models.

(Best) Example: Compilers
Model transformation and the design process

Source: Daniel Varro, CSMR2012
Approach 3: Multi-aspect modeling

A *system aspect*, or *system view*, is a way to look at or describe a system as a whole. Each system aspect has its own associated semantic domain and can provide an exhaustive description of the system, but only from that particular point of view.
System-level Modeling

Examples

Methods war!

Source: Emertxe Ltd
Advantages of multi-aspect modeling

System-level Modeling

- **Tame complexity**
  - Deal with one aspect at a time
  - Every aspect contributes to one system model

- **Co-modeling**
  - Interdisciplinary design trade-offs
  - Tooling profits from multi-domain information

- **Productivity**
  - Increased (re)usability of models
  - More flexible evaluation of design alternatives

**Challenges**

- Explicit (formal?) interdependency between aspects
- Neoteric (new + isoteric) views
An example from CERBERO 1/3

Task aspect

Behavior aspect
An example from CERBERO 2/3

Physical aspect

System-level Modeling

Karol Desnos (IETR) & Julio Oliveira (TNO)
An example from CERBERO 3/3

Mapping view
System-level Modeling

Using the models together to assess KPIs

- task model
- behavioural model
- physical model
- $T$ $P$ mapping
  - temporal behaviour
  - power requirement
  - dependability
  - <other performance indicators>

feedback

compare

requirements
  - temporal behaviour
  - power requirement
  - dependability
  - <other perf. indicators>
System-level Modeling

Some CERBERO contributions (expected)

Explicit (formal?) interdependency between aspects

- Research: Aspects can also be used to formalize the dependency between other aspects.
- Research: A (semantic) intermediary representation layer as connection point between modeling aspects (modeling languages)

Challenges

Neoteric (new + isoteric) views

- Research: How to model reconfigurability? Adaptivity? Scalability and Cardinality?
- Research: How to model key performance indicators in a more formal way?
Component level > Outline

Component level
SW/HW (co-)design

1. State-of-the-Art
2. Models of Computation
3. Models of Architecture
Component level > State of the Art

**Need for a new HW/SW design approach!**

- **Lines of code/chip**: x2 every 10 months
- **Transistors/chip**: x2 every 18 months
- **Lines of code/day**: x2 every 5 years

**Software Productivity Gap**

Source: ITRS & *Hardware-dependent Software*, Ecker et al., Springer
Typical HW/SW component

Heterogeneous Multiprocessor System-on-Chip (MPSoC)

- Customized Processors for Specific Functions
- "Generalist" Processors
- Reconfigurable Logic
- On-Chip Interconnect(s)
- Globally Shared Storage Space
- Interfaces with External World
Component level > State of the Art

Typical development flow

C Code

> Command line options

Compiler

10010 Binaries

Simulator + Debugger + Profiler

OS

PEs Core(s)

Bitstream / mask

HDL

Synthesizer

Place & Route
C Language is:

- Good for abstracting core architecture
  - Amount of registers
  - Number of pipeline stages
  - Instruction parallelism

- Bad for expressing coarse-grain parallelism
  - Inspired by Turing Machine
  - Global state in a program
VHDL/Verilog Languages are:

- **Good for abstracting**
  - Transistors
  - Analog concerns (signal propagation time)

- **Bad for abstracting**
  - Software concerns
  - ... (more reasons in HLS and HW courses)
Component level > State of the Art

What we want?

- High-level desc. Algorithm
- High-level desc. Architecture
- Cross-Layer DSE tool
- Simulator + Debugger + Profiler
- Portable Program

Karol Desnos (IETR) & Julio Oliveira (TNO)
Component level > Outline

Component level
SW/HW (co-)design

1. State-of-the-Art
2. Models of Computation
3. Models of Architecture

Model of Computation (MoC)

*a.k.a. programming paradigm*

**Definition:**
- A set of operational elements that can be composed to describe the behavior of an application.

→ *Semantics of the MoC*

**Objective:**
- Specify implementation-independent system behavior.
- Ease specification, implementation, verification of system properties.

**How:**
- MoCs act as the interface between computer science & mathematical domain.

⚠️ A MoC is not a language ⚠️
Language

Definition:
• A set of textual/graphical symbols that can be assembled respecting a well defined grammar to specify the behavior of a program.

→ Syntax of a the language

Objective:
• Ease system description and maximize developer productivity.
• Be developer-friendly: readability, reusability, modularity, …

How:
• Languages are the interface between the programmer & the Machine (through the compiler).

A Language implements one or several MoCs
MoC Semantics and Language Syntax

- UML implements object-oriented semantics
- C++/Java implements object-oriented semantics
- They share semantics but not syntax

UML

```
BankAccount
  owner : String
  balance : Dollars

strike ( amount : Dollars )
withdrawal ( amount : Dollars )
```

Java

```
public class SavingsAccount
  extends BankAccount {
    private int annualInterestRate;
    public void withdrawal(int v){
      ...
    }
  }
```

Component level > MoCs
A few MoCs

Finite State Machine MoCs

Semantics
- States
- Transitions (possibly conditional)

Used for
- Sequential logic
- System-level behavior
- Communication protocols
- ...

Property
- Non-deterministic, sequential

Glass FSM

Pour beverage
Drink
Be sad

Empty
Full
Broken
A few MoCs

Petri Nets

Semantics
• Places
• Transitions & Arcs

Used for
• Synchronization protocols
• Parallel computations
• ...

Property
• Parallelism
• Liveness, Boundedness, Reachability
A few MoCs

Discrete Event MoCs

Semantics
- Modules
- Signals
- Timed events
- Global clock

Used for
- Hardware Description
- “System” Simulation

Properties
- Timed, Non-deterministic (if badly used)
A few MoCs

Kahn Process Network

Semantics
• Actors & ports
• FIFO queues

Used for
• Parallel computations
• Stream processing

Properties
• Deterministic
• Untimed

A few MoCs

Kahn Process Network (KPN)

Determinism

Process sort(in int i, out int even, out int odd) {
    int value = i.read(); // Blocking
    if( value % 2 == 0)
        even.write(value);
    else
        odd.write(value);
}
A few MoCs

Kahn Process Network (KPN)

Determinism

```c
Process interleave(in int a, in int b, out int o) {
    static bool = true;
    int value = (bool)? a.read() : b.read();
    bool = !bool;
    o.write(value);
}
```
A few MoCs

Dataflow Process Network (DPN)

Non-Determinism

Process intleleave(in int a, in int b, out int o) {
    static bool = true;
    int value = (bool)? a.read() : b.read();
    bool = !bool;
    if(no_timeout) o.write(value);
}
A few MoCs

Synchronous Dataflow

Semantics
• Actors & ports
• FIFO queues

Used for
• Parallel computations
• Stream processing

Properties
• Liveness
• Boundedness
• Deterministic
• Untimed

MoC Properties are important.

You need to know them to select the MoC suiting your needs.

<table>
<thead>
<tr>
<th>Feature</th>
<th>SDF</th>
<th>ADF</th>
<th>IBSDF</th>
<th>DSSF</th>
<th>PSDF</th>
<th>PiSDF</th>
<th>SADF</th>
<th>SPDF</th>
<th>DPN</th>
<th>KPN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expressivity</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hierarchical</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Compositional</td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reconfigurable</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Statically schedulable</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Decidable</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>(X)</td>
<td>(X)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Variable rates</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Non-determinism</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

SDF: Synchronous Dataflow
ADF: Affine Dataflow
IBSDF: Interface-Based Dataflow
DSSF: Deterministic SDF with Shared Fifos
PSDF: Parameterized SDF
PiSDF: Parameterized and Interfaced SDF
SADF: Scenario-Aware Dataflow
SPDF: Schedulable Parametric Dataflow
DPN: Dataflow Process Network
KPN: Kahn Process Network
Component level > Outline

Component level
SW/HW (co-)design

1. State-of-the-Art
2. Models of Computation
3. Models of Architecture
Models of Architecture

Definition:
• Formal representation of the operational semantics of networks of functional blocks describing architectures.

Abstraction
• VHDL/Verilog
• SPIRIT IP-XACT
• SystemC TLM
• UML Marte
• AADL
• S-LAM

Disclaimer:

The topic of MoA is not as extensively covered in the scientific literature/industrial tools.

Ideas presented in following slides are based on recent work by Pelcat et al.

Unfortunately, these ideas can not (yet) be considered as a globally accepted reference.

Models of Architecture

Definition:
• Formal representation of the operational semantics of networks of functional blocks describing architectures

Yes, but
• The SDF MoC is a formal representation of the operational semantics of networks of functional blocks describing applications
• What if application = architecture?
• What if we do not want to model the architecture as a network?
• We need a more precise definition for MoAs!
Models of Architecture

Definition

• an abstract *efficiency model* of a system architecture that provides a unique, reproducible cost computation, when processing an application described with a specified MoC.

Algo G conforms to MoC

Archi H conforms to MoA

Deployment

One and always the same performance number
**LSLA MoA Example**

**Linear System Level Architecture (LSLA)**

- A Model of Architecture
- Computing additive costs from application activity
- That can be used, for instance, to predict energy

![Diagram of LSLA MoA Example](image)
LSLA MoA Example

Application Activity for cost computation

– AA is the amount of effort to execute an application
– From the MoC, we derive

• processing and communication tokens (e.g. tasks and messages)
• processing and communication quanta (e.g. cycles and Bytes)
Component level > MoAs

LSLA MoA Example

Cost = \(2 \times (2x+1) + 1 = 5\)

Cost = \(4 + 2 + 20 + 5 + 12 + 3 + 3 + 8 = 57\)
LSLA MoA example

Experiments
- On an ARM big LITTLE architecture
- Running a stereo matching application
- A simple LSLA reaches 85% of fidelity
Models

• offer various levels of abstraction for designing system-of-systems, system, SW and HW components

• are backed by mathematical models, providing a base for verification of system properties

• are based on semantics can be translated into actual implementation. (cf. next lecture)