Open SystemC Initiative
AMS Draft 1 Standard

Martin Barnasconi
AMS Working Group Chair
September 8, 2009

Overview

- Motivation and requirements
  - Why having AMS extensions for SystemC?
  - Positioning of SystemC AMS extensions
  - Requirements
- Open SystemC Initiative (OSCI)
- Design refinement methodology
  - Modeling formalisms and use cases
  - Abstraction levels
- SystemC AMS extensions - Draft 1 Standard
  - AMS Draft 1 kit contents
  - Execution semantics and language constructs
  - What's new?

Motivation and requirements

Why having AMS extensions for SystemC?

- Missing is
  - An agreed system modeling language and methodology to design embedded AMS systems
  - An open modeling and programming interface between AMS and digital HW/SW system descriptions
  - A platform that facilitates AMS model exchange and reuse of intellectual property (IP)
  - An architecture design tool for AMS system-level design and verification

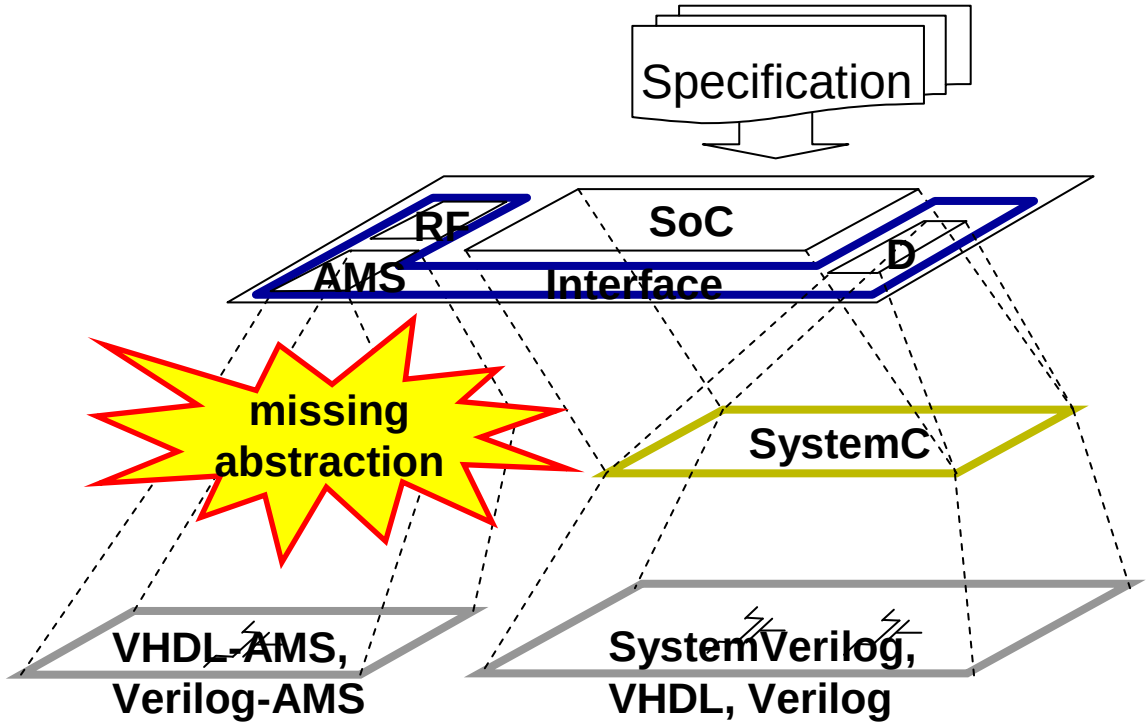
- It's time to standardize *AMS extensions* for SystemC !
  - Open SystemC Initiative (OSCI) will drive standardization, deployment and support of the SystemC AMS extensions
  - Targeting an open source standard for system-level design for Embedded AMS systems

Positioning SystemC AMS Extensions

Functional

Architecture

Implementation

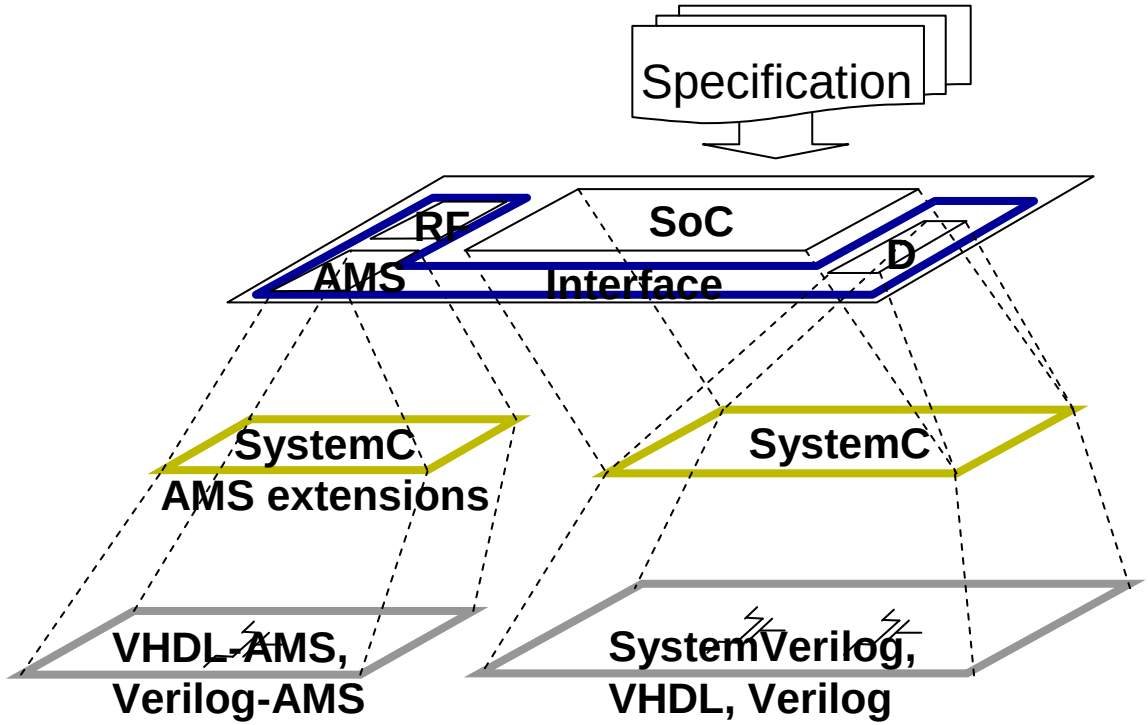


Positioning SystemC AMS Extensions

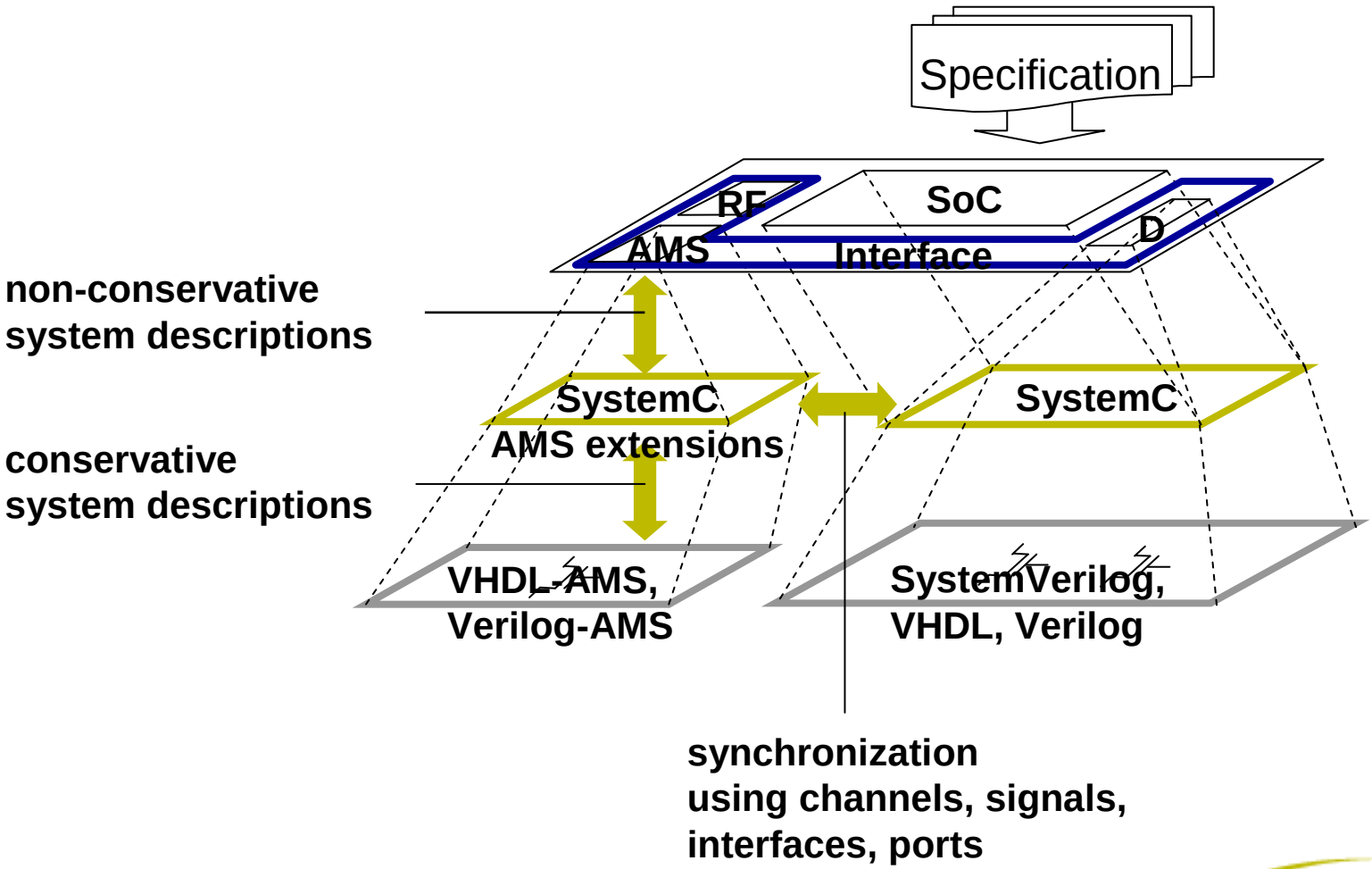
Functional

Architecture

Implementation



Positioning SystemC AMS Extensions



Requirements

- Design of True Heterogeneous Systems-on-a-chip
  - Analog, Mixed-signal, RF, digital HW/SW interaction
  - Telecom: high frequency, high bandwidth, configurable radio's
- Support different levels of design abstraction
  - Functional modeling, architecture design, circuit implementation
- Support different use cases
  - Executable specification, architecture exploration, virtual prototyping, integration validation
- Need for Virtual Prototype Platform including both digital HW/SW and AMS/RF system-level modeling

Open SystemC Initiative (OSCI)

What is OSCI?

- The Open SystemC Initiative (OSCI) is
  - An independent not-for-profit organization
  - Composed of a broad range of companies, universities and individuals
  - Dedicated to deploy and support SystemC as an open source standard for system-level design

- SystemC provides
  - Hardware-oriented constructs as a class library implemented in C++
  - An interoperable modeling platform for development and exchange of C++ models
  - A stable platform for development of system-level tools
  - Models for design and verification, from concept to implementation in hardware and software

OSCI Members and key contributors

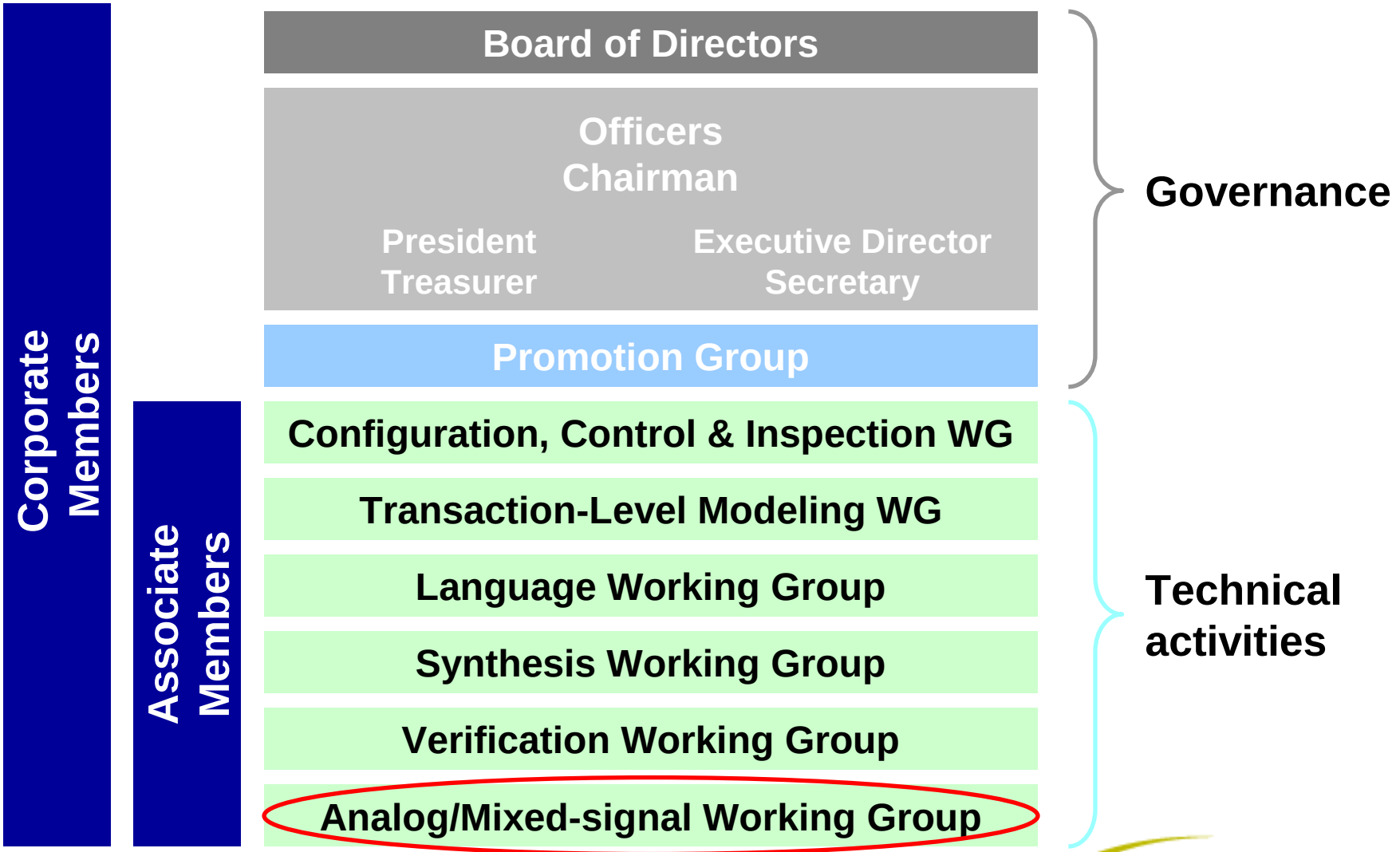
Corporate Members

Associate Members

Affiliate Members

SYSTEMC™

OSCI Organization and Working Groups



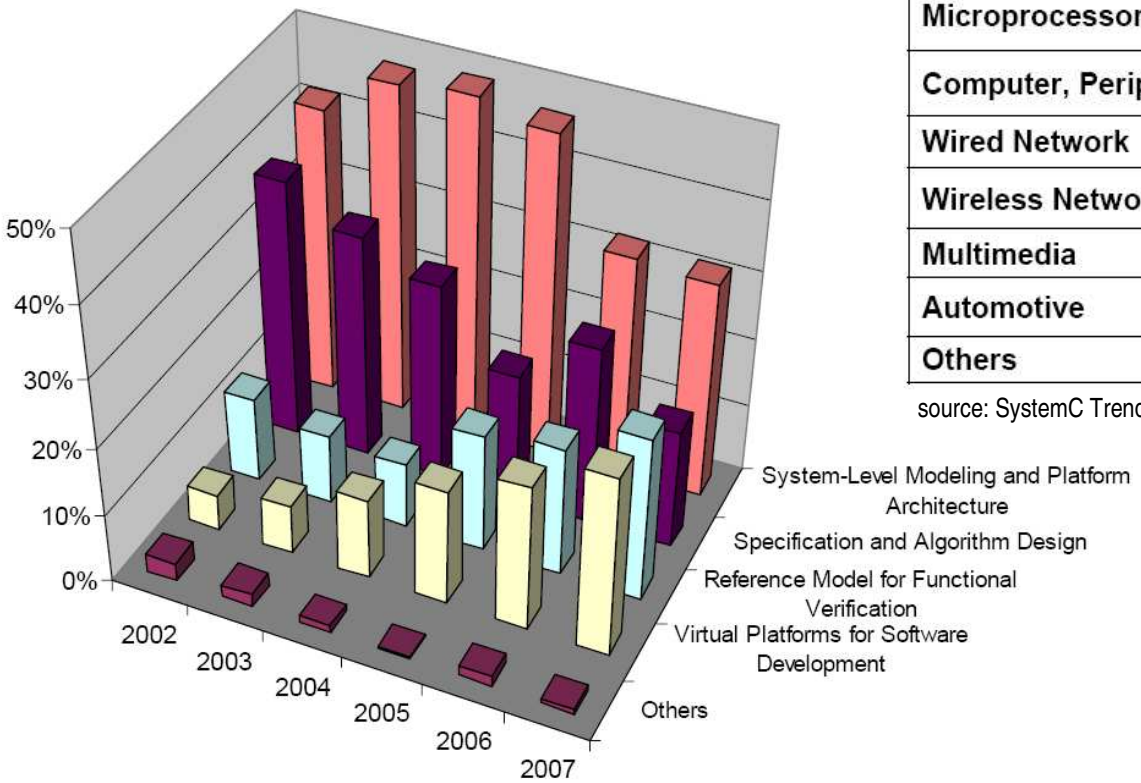
Current OSCI AMS Working Group Roster

- Steady growth in AMS WG: 53 individuals from 18 organizations
  - strong drive from semiconductor industry
  - full support of universities and research institutes
  - growing interest and participation of EDA/ESL vendors

- Chair: Martin Barnasconi, NXP Semiconductors
  Vice chair: Christoph Grimm, Vienna University of Technology

SystemC User Group Survey Report

- SystemC use cases



- End Product Market Segments

End Product Markets	2003	2004	2005	2006	2007
Microprocessor/DSP	18.9%	16.0%	13.1%	10.5%	14.7%
Computer, Peripheral	22.9%	21.6%	18.5%	24.2%	19.0%
Wired Network	11.2%	5.2%	5.8%	4.8%	5.2%
Wireless Network	13.1%	10.4%	13.1%	7.3%	6.9%
Multimedia	25.6%	34.2%	33.8%	37.9%	31.9%
Automotive	1.9%	3.0%	3.8%	4.0%	4.3%
Others	6.4%	9.7%	11.9%	11.3%	18.1%

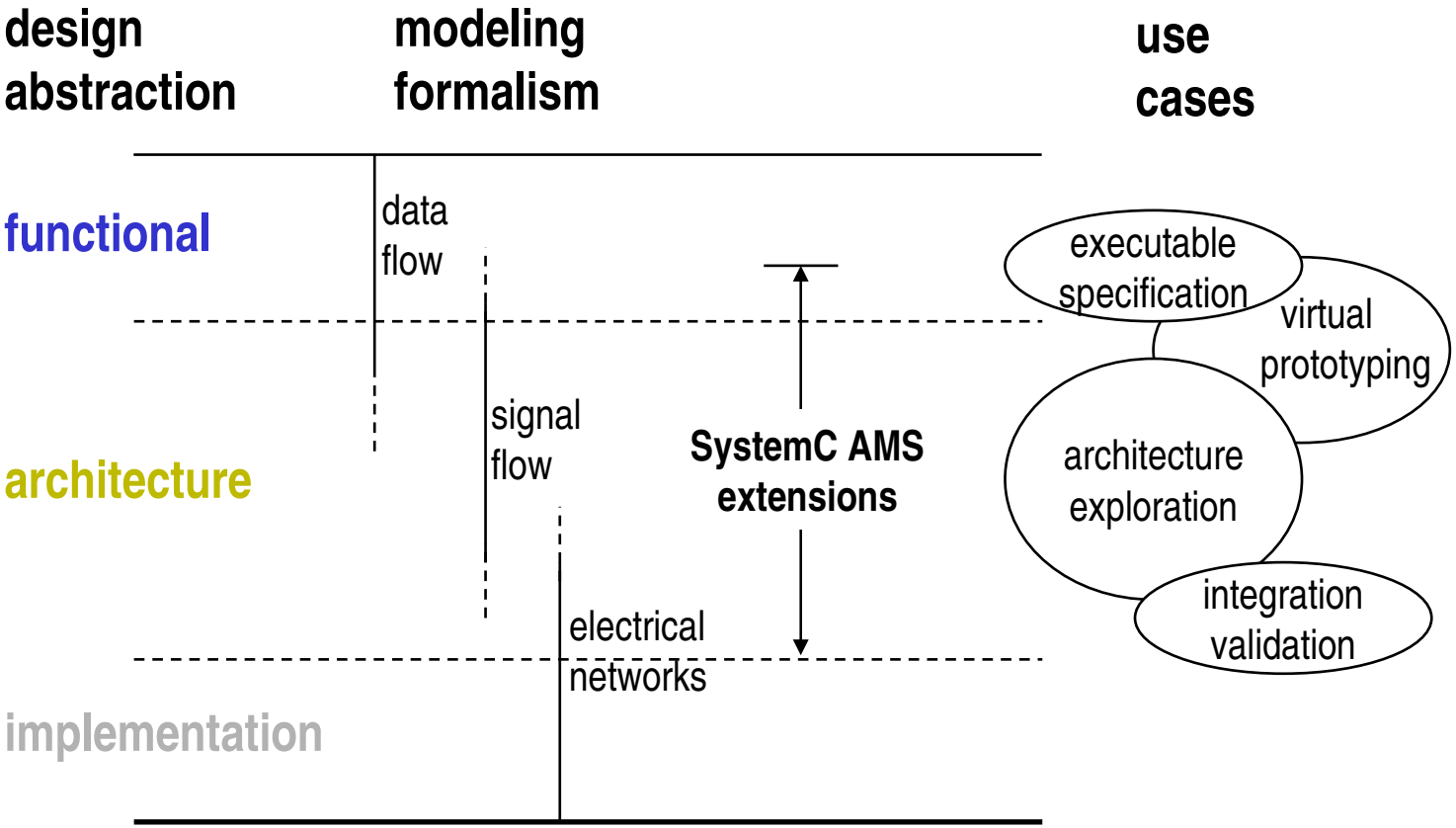
source: SystemC Trends report, April 2007

focus of OSCI AMS WG

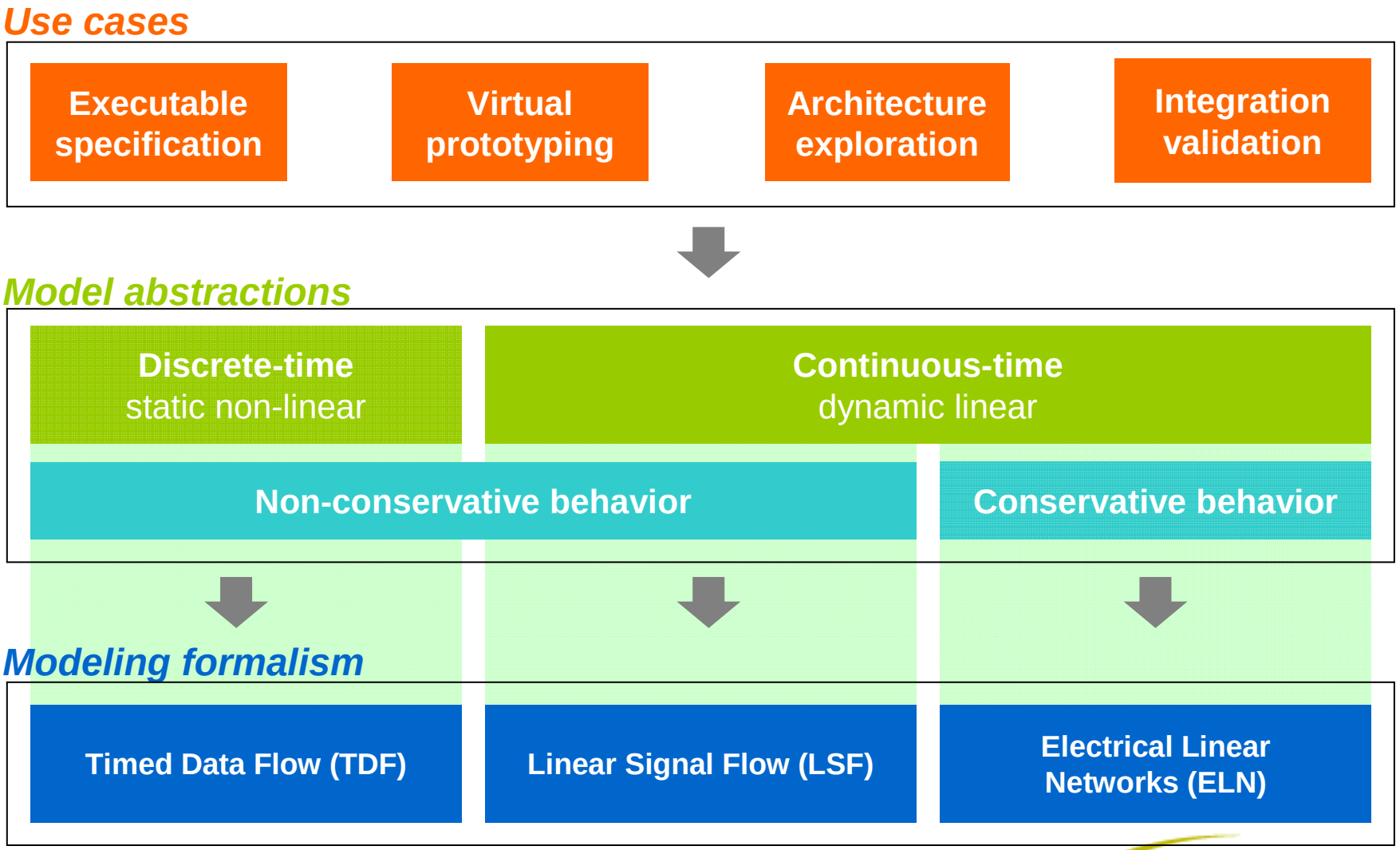
- Full document available on www.systemc.org

Design refinement methodology

Modeling formalisms and use cases



Model abstraction and formalisms

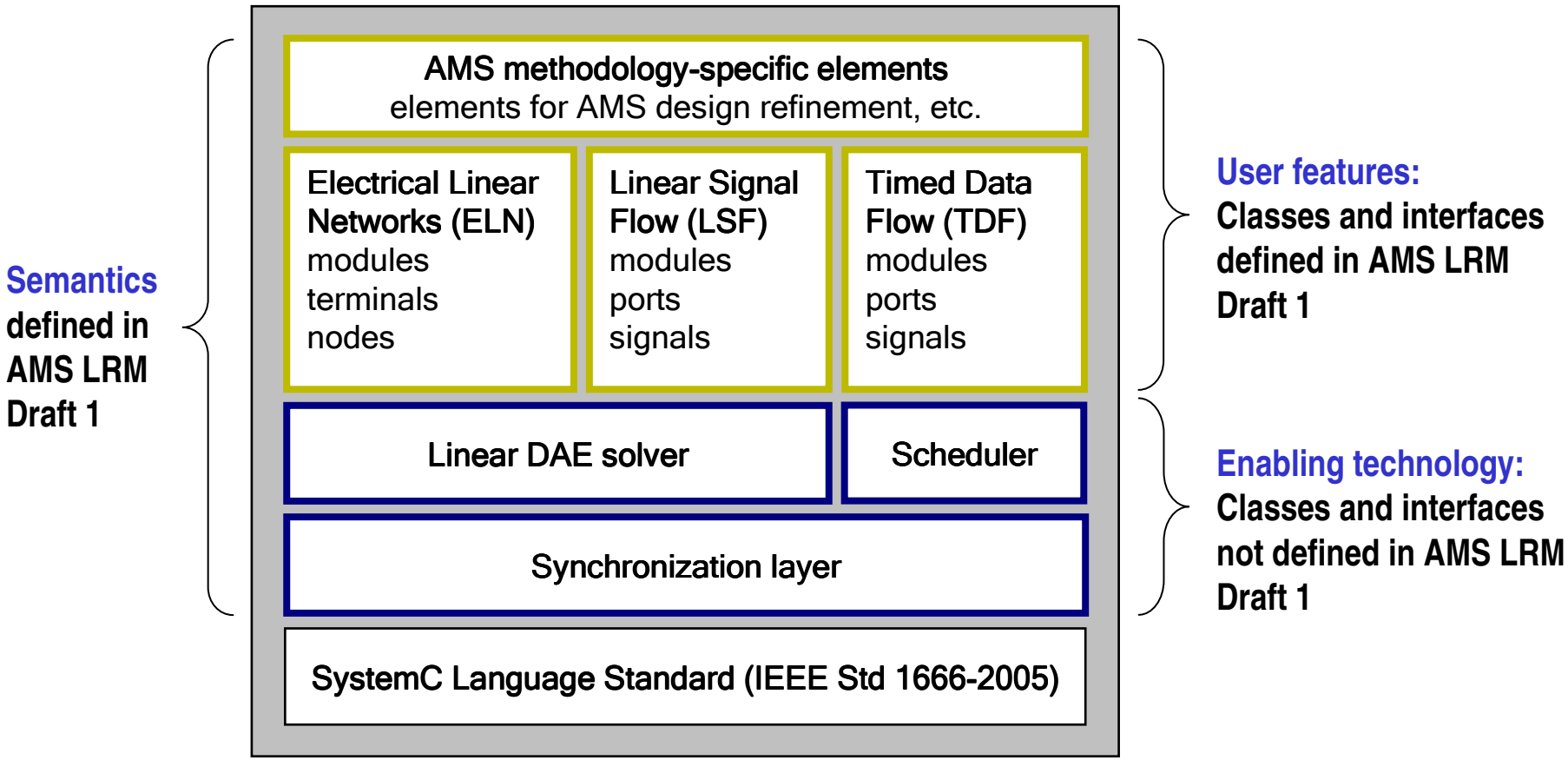


AMS Draft 1 Standard

AMS Draft 1 standard – kit contents

- AMS Draft 1 standard has been released in December 2008
  - Preparation for “AMS 1.0” Standard ongoing
- AMS Draft 1 kit contents
  - Draft Standard SystemC AMS extensions Language Reference Manual
  - Requirements specification for SystemC AMS extensions
  - Whitepaper “An Introduction to Modeling Embedded Analog/Mixed-Signal Systems using SystemC AMS extensions”
  - Code example SystemC AMS extensions
- Available for download on www.systemc.org

SystemC AMS extensions LRM Draft 1



SystemC AMS extensions – concept 1/3

- AMS modeling formalisms based on known *models of computation* (MoC)
  - Data flow
  - Signal flow
  - Electrical networks
- AMS models of computation are not based on communication / synchronization of processes
  - instead, AMS descriptions represent an *equation system*
- An AMS primitive module represents a set of equation, which has to be contributed to the overall equation system
- An AMS interface / channel represents a node in a conservative system or a variable in a non-conservative system

SystemC AMS extensions – concept 2/3

- Timed Data Flow - efficient simulation of discrete-time behavior
  - Data flow simulation accelerated using static scheduling
  - Schedule is activated in discrete time steps, introducing timed semantics
  - Support of static non-linear behavior
- Linear Signal Flow - simulation of continuous-time behavior
  - Differential and Algebraic Equations solved numerically at appropriate time steps
  - Pre-defined LSF primitive modules for adders, integrators, differentiators, transfer functions, etc.

not available in existing prototype

- Electrical Linear Networks - simulation of network primitives
  - Network topology results in equation system which is solved numerically
  - Pre-defined ELN primitive modules for linear components (e.g. resistors, capacitors) and switches

SystemC AMS extensions – concept 3/3

- AMS methodology-specific elements
  - Unified design refinement methodology to support different use cases
  - Time domain simulation and Small-signal frequency-domain AC and noise analysis
- User-defined AMS extensions
  - Additional simulators and solvers can be linked in a C++ manner
  - Using the synchronization layer for the communication with SystemC
- Synchronization with SystemC
  - Fixed time-step synchronization with SystemC
  - Predefined converter ports and converter modules/primitives to synchronize between TDF, LSF and/or ELN and SystemC
- Each *model of computation* has its own namespace
  - Timed Data Flow: sca_tdf
  - Linear Signal Flow: sca_lsf
  - Electrical Linear Networks: sca_eln

not available in existing prototype

SystemC AMS extensions - module types

- AMS modules are derived from sca_core::sca_module which is derived from sc_core::sc_module
  - note: not all sc_core::sc_module member functions can be used
- AMS modules are always *primitive* modules
  - an AMS module can not contain other modules and/or channels
- Hierarchical descriptions still use sc_core::sc_module (or SC_MODULE macro)
- Depending on the MoC, AMS modules are pre-defined or user-defined
- Language constructs
  - for TDF: sca_tdf::sca_module (or SCA_TDF_MODULE macro)
  - for LSF and ELN: instantiate pre-defined primitive modules directly

SystemC AMS extensions - channel types

- AMS channels are derived from sc_interface
- AMS channels for Time Data Flow and Linear Signal Flow
  - Based on directed connection
  - Used for non-conservative AMS model of computation
  - Language constructs
    - sca_*MoC*::sca_signal
    - e.g. sca_lsf::sca_signal, sca_tdf::sca_signal<T>
- AMS channels for Electrical Linear Networks
  - Conservative, non-directed connection
  - Characterized by an across (voltage) and through (current) value
  - Language constructs
    - sca_*MoC*::sca_node / sca_*MoC*::sca_node_ref
    - e.g. sca_eln::sca_node, sca_eln::sca_node_ref

TDF language constructs

- Predefined classes
  - sca_tdf::sca_module
  - sca_tdf::sca_signal_in_if
  - sca_tdf::sca_signal_out_if
  - sca_tdf::sca_signal
  - sca_tdf::sca_in
  - sca_tdf::sca_out
  - sca_tdf::sc_core::sc_in (sca_tdf::sc_in)
  - sca_tdf::sc_core::sc_out (sca_tdf::sc_out
- (some) member functions
  - set_delay, get_delay
  - set_rate, get, rate
  - set_timestep, get_timestep
  - set_timeoffset
  - read, write
  - kind
  - set_attributes
  - initialize
  - processing
  - ac_processing
  - ...

Example: TDF language constructs

```
SCA_TDF_MODULE(mytdfmodel)              // create your own TDF primitive module
{
  sca_tdf::sca_in<double>  in1, in2;  // TDF input ports
  sca_tdf::sca_out<double> out;       // TDF output port

  void set_attributes()
  {
    // placeholder for simulation attributes
    // e.g. time step between module activations
  }

  void initialize()
  {
    // put your initial values here
  }

  void processing()
  {
    // put your signal processing or algorithm here
  }

  SCA_CTOR(mytdfmodel) {}
};
```

LSF language constructs

not available in existing prototype

- Predefined classes
  - sca_lsf::sca_in
  - sca_lsf::sca_out
  - sca_lsf::sca_signal
  - sca_lsf::sca_add
  - sca_lsf::sca_sub
  - sca_lsf::sca_gain
  - sca_lsf::sca_dot
  - sca_lsf::sca_integ
  - sca_lsf::sca_delay
  - sca_lsf::sca_source
  - sca_lsf::sca_ltf_nd
  - sca_lsf::sca_ltf_zp
- Predefined classes (cont.)
  - sca_lsf::sca_ss
  - sca_lsf::sca_tdf::sca_source
  - sca_lsf::sca_tdf::sca_gain
  - sca_lsf::sca_tdf::sca_mux
  - sca_lsf::sca_tdf::sca_demux
  - sca_lsf::sca_tdf::sca_sink
  - sca_lsf::sc_core::sca_source
  - sca_lsf::sc_core::sca_gain
  - sca_lsf::sc_core::sca_mux
  - sca_lsf::sc_core::sca_demux
  - sca_lsf::sc_core::sca_sink
  - ...

Example: LSF language constructs

not available in existing prototype

```
SC_MODULE(mylsfmodel)          // create a model using LSF primitive modules
{
  sca_lsf::sca_in  in;         // LSF input port
  sca_lsf::sca_out out;        // LSF output port

  sca_lsf::sca_signal sig;     // LSF signal

  mylsfmodel(sc_module_name, double fc=1.0e3) // Constructor with
  {                                           // parameters

    sub1 = new sca_lsf::sca_sub("sub1");      // instantiate predefined
    sub1->x1(in);                             // primitives here
    sub1->x2(sig);
    sub1->y(out);

    dot1 = new sca_lsf::sca_dot("dot1", 1.0/(2.0*M_PI*fc) );
    dot1->x(out);
    dot1->y(sig);
  }

};
```

ELN language constructs

- Predefined classes
  - sca_eln::sca_terminal
  - sca_eln::sca_node
  - sca_eln::sca_node_ref
  - sca_eln::sca_r
  - sca_eln::sca_l
  - sca_eln::sca_c
  - sca_eln::sca_vcvs
  - sca_eln::sca_vccs
  - sca_eln::sca_ccvs
  - sca_eln::sca_cccs
  - sca_eln::sca_nullor
  - sca_eln::sca_gyrator
  - ...
- Predefined classes (cont.)
  - sca_eln::sca_vsource
  - sca_eln::sca_vsink
  - sca_eln::sc_tdf::sca_vsource
  - sca_eln::sca_tdf::sca_isource
  - sca_eln::sc_core::sca_vsource
  - sca_eln::sc_core::sca_isource
  - sca_eln::sca_tdf::sca_r
  - sca_eln::sca_tdf::sca_l
  - sca_eln::sca_tdf::sca_c
  - sca_eln::sc_core::sca_r
  - sca_eln::sc_core::sca_l
  - sca_eln::sc_core::sca_c
  - ...

Example: ELN language constructs

```
SC_MODULE(myelnmodel)              // model using ELN primitive modules
{
  sca_eln::sca_terminal in, out; // ELN terminal (input and output)

  sca_eln::sca_node_ref gnd;     // ELN reference node

  SC_CTOR(myelnmodel)              // standard constructor
  {
    r1 = new sca_eln::sca_r("r1", 10e3);  // instantiate predefined
    r1->p(in);                            // primitive here (resistor)
    r1->n(out);

    c1 = new sca_eln::sca_c("c1", 100e-6);
    c1->p(out);
    c1->n(gnd);
  }

};
```

Summary

- Objectives of the SystemC AMS extensions
  - An open source language standard for system-level design and verification dedicated to analog/mixed-signal systems
  - Support a variety of use cases: executable specification, architecture exploration, integration validation
  - Defines a unified AMS design refinement methodology using data flow, signal flow and electrical networks modeling formalisms
- SystemC AMS standardization effort in OSCI
  - AMS Draft 1 standard available on www.systemc.org
  - Introducing AMS execution semantics and language constructs
  - Preparation for “AMS 1.0” Standard ongoing
- Get involved and join the AMS forum on www.systemc.org

More information

- www.systemc.org
- www.systemc-ams.org

Thank You