# Grand Challenges in Modeling and Simulation: What Can DEVS Theory Do To Meet Them? Parts 1 and 2

**Seminar to School of Automation Science and Electrical Engineering, Beihang University, Beijing, China**

http://ev.buaa.edu.cn/

December, 2013

**Bernard P. Zeigler, Zeigler@rtsync.com**
**Emeritus Professor Univ. Arizona**
**Chief Scientist**
**RTSync Corp**

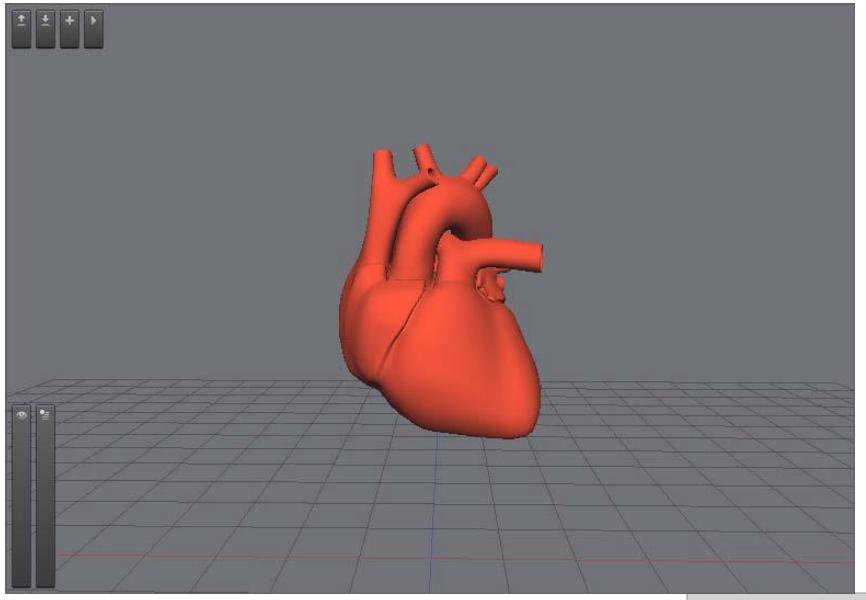
**A recent panel at the Winter Simulation Conference listed grand challenges for modeling and simulation research.** [*]

- **In this seminar, we will consider such challenges in the context of the Theory of Modeling (TMS) and Simulation and the Discrete Event System Specification (DEVS) formalism.**
- **After introducing TMS and DEVS, we will examine individual challenges and ask what can TMS and DEVS do to address them.**
- **If insufficient, what more is needed.**
- **State-of-the art results by DEVS and other researchers will be brought in to illuminate the discussion.**

[*] **CHALLENGES FOR MODELING AND SIMULATION Simon J. E. Taylor ,Paul A. Fishwick, Richard Fujimoto Adelinde M. Uhrmacher, Ernest H. Page , Gabriel Wainer Proceedings of the 2012 Winter Simulation Conference, C. Laroque, J. Himmelspach, R. Pasupathy, O. Rose, and A. M. Uhrmacher, eds.**

# Part 1 Challenges and Introduction to DEVS

# Fishwick's Challenge: 3 Models of the Same Thing



computer graphics geometry **model**

Ontology **model**

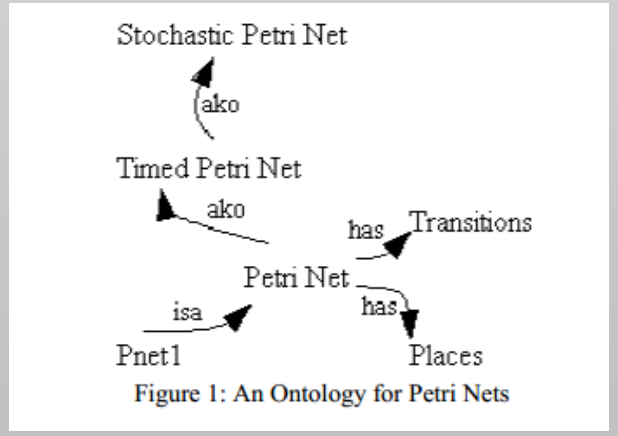
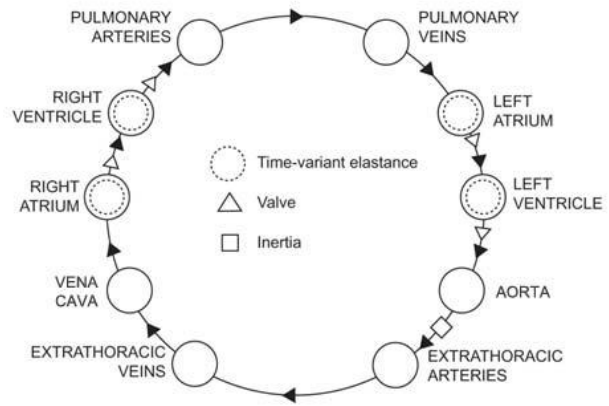


Figure 1: An Ontology for Petri Nets

Simulation **model** captures dynamics of blood flow




$$p(t) = E * (v(t) - UV)$$

$$f(t) = \frac{p_{in}(t) - p(t)}{R}$$

$$\frac{dv}{dt}(t) = f(t) - f_{out}(t)$$

$E$ : elastance
$f$ : flow
$p$ : pressure
$R$ : resistance
$v$ : volume
$UV$ : unstressed volume

**Ezzell, Z., P. A. Fishwick, B. Lok, S. Lampotang, and A. Pitkin. 2011. "An Ontology-Enabled User Interface for Simulation Model Construction and Visualization." *Journal of Simulation*, 5 (3), 147-156.**

# Fishwick's Challenge Cont'd

**These models described and demonstrate heat function**

**Here are some questions :**

- **How do we formalize each of the models? What information structures do we employ?**
- **How are the model components interconnected using these formalisms?**
- **How can we navigate multiple abstraction levels of each of the models?**
  - **The simulation model is based on ordinary differential equations,**
  - **but a lower-level of detail model might employ finite element analysis (for structural questions)**
  - **or fluid mechanics (for blood flow). Scales across space and time require different model structures.**
- **How do we connect each of these models with actual hearts?**

# Enders Game (the movie): Blending Simulation Models with Reality *

**You can solve the integration problem using software tools for once-only solutions or using theory for general solution**

**“The concept of Integrative modeling is to connect simulation models, expressed mathematically or through diagrams, with real and virtual objects and environments. By allowing models to be fully integrated within the human-model interface, concepts and objects are more tightly interconnected, much like a phrase is hyperlinked to a target document.”**

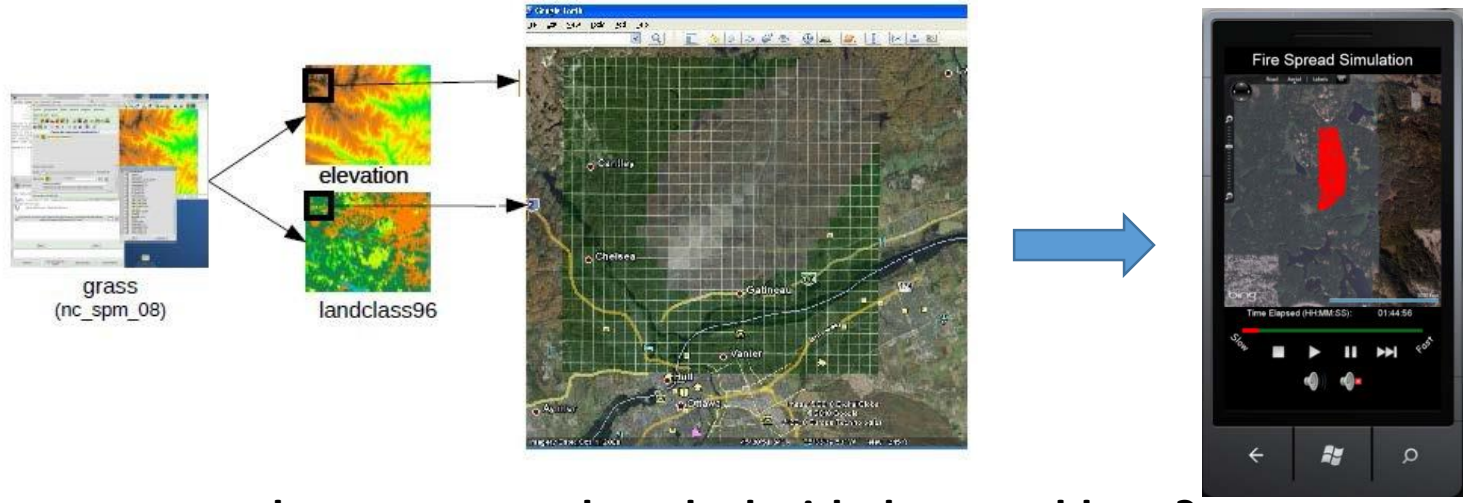
*** Paul Fishwick http://www.cise.ufl.edu/~fishwick/Research.html**

# Uhrmacher's Challenge

- **Certain spatial phenomena like molecular crowding can only be analyzed at higher level of detail**
- **Many different expressive languages for spatial modeling and simulation are being developed.**
- **How do we evaluate the impact of new simulation algorithms**
  - **select the most adequate one for the task at hand, and**
  - **combine them so that crucial model parts are still executed in sufficient detail?**
- **How can the process of evaluation be standardized and automated?**
- **How can the user be informed about trading accuracy for speed?**

# Wainer's Challenge

- **How to interface simulation software with Smartphone APIs?**
- **How to combine discrete-event simulation, cloud computing (with web services interfaces) and mobile devices for distributed simulation and collaboration?**
- **How to build advanced mashup applications using simulation, sharing and reusing models and experiments?**




- **How to use a more abstract approach to deal with these problems?**
  - **instead of dealing with the data and software levels, interoperability will be dealt with at the modeling and experimentation levels,**
  - **Improving reuse and providing better ways to mashup models, experiments and other services**

# Wainer's Challenge Cont'd

- **We need to explore new architectures, including hybrid approaches suitable for handheld devices.**
- **We need to explore new mechanisms to allow users to collaborate on joint simulation exercises, sharing information and data between existing devices and the simulation server.**

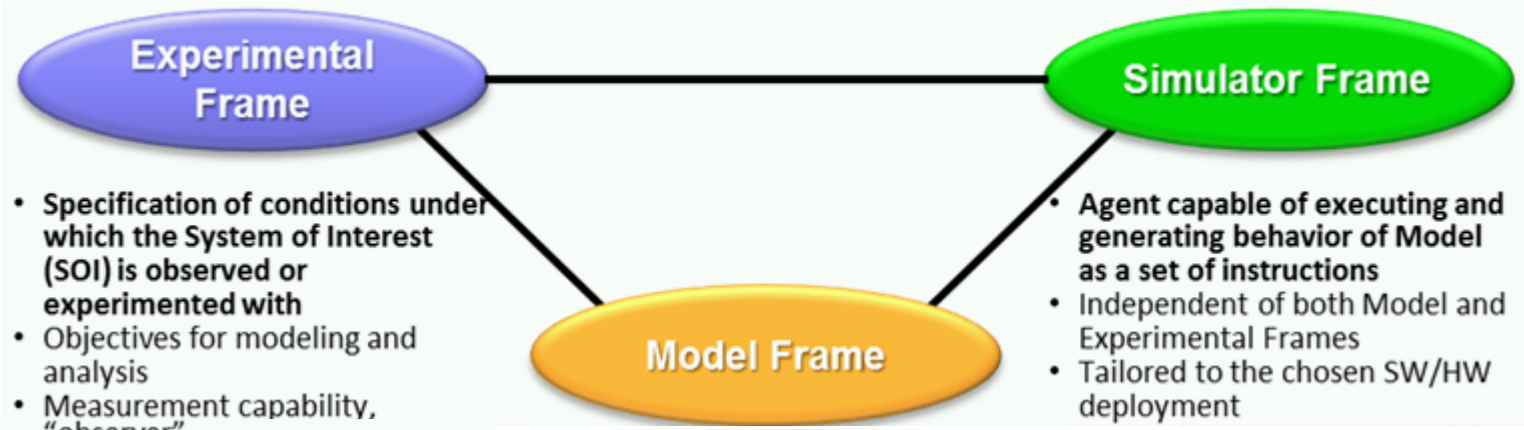
- **Formal M&S techniques like DEVS (Discrete Events Systems Specification), promise better success by addressing these issues at a higher level of abstraction**
- **With DEVS, models, simulators and experiments are systematically built and interoperability can be enhanced.**

# Brief Introduction to DEVS

# Overview

- Review of  Systems M&S Framework and DEVS formalism
- DEVS Environments and Applications
- Example  of Model Development
- DEVS unique features – multiformalism, dynamic structure, standardization

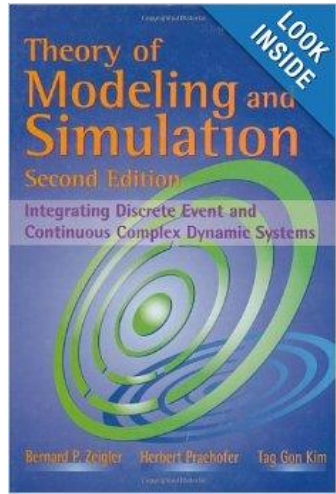
# DEVS-Based Framework For Systems Modeling and Simulation



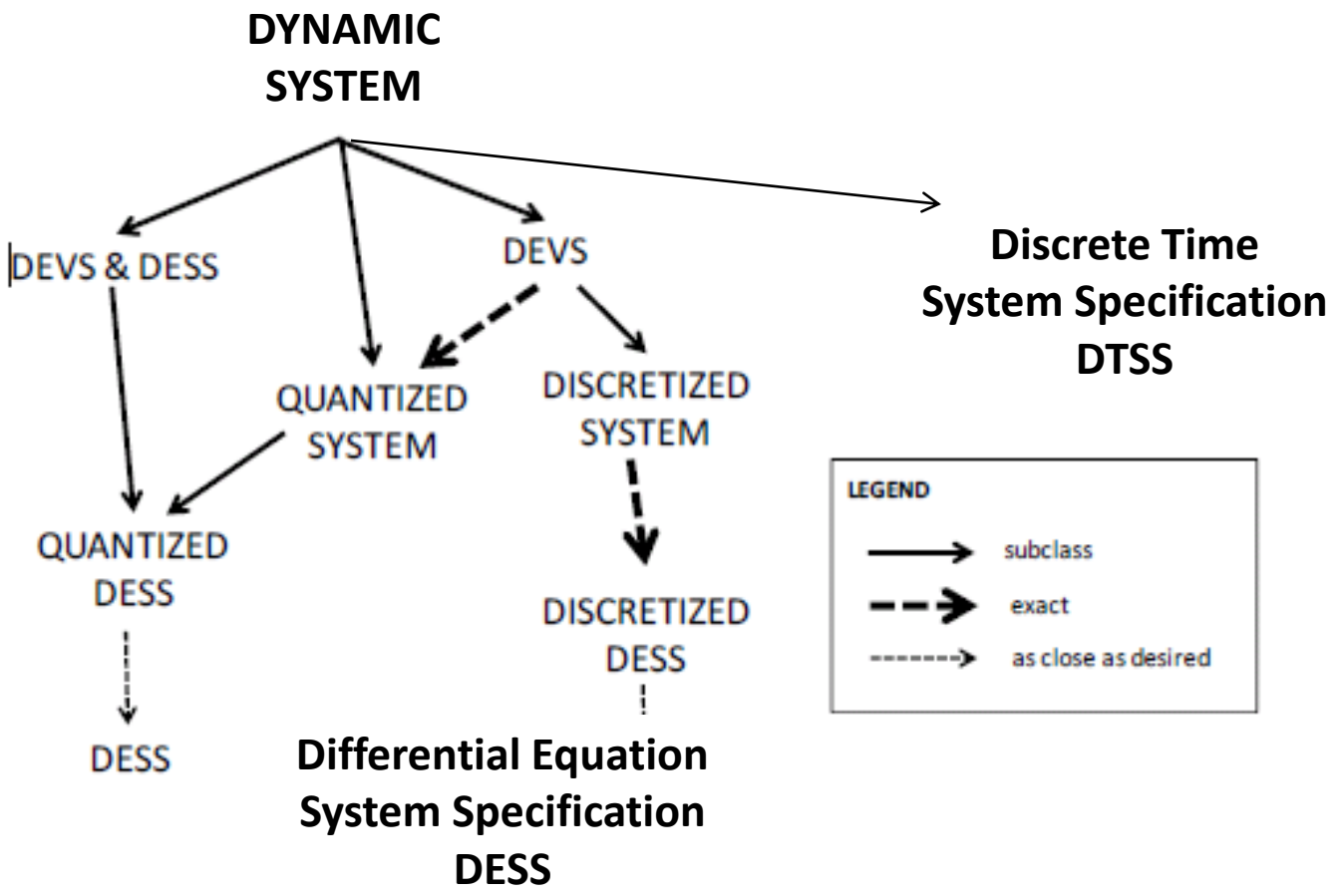

- *Explicit rules for expressing an Experimental Frame in the DEVS Reference Architecture*

*Explicit rules for expressing a Model in the DEVS Reference Architecture*

- *Correctly executes any Model and Experimental Frame constructed to the DEVS Reference Architecture*



# DEVS – Discrete Event System Specification



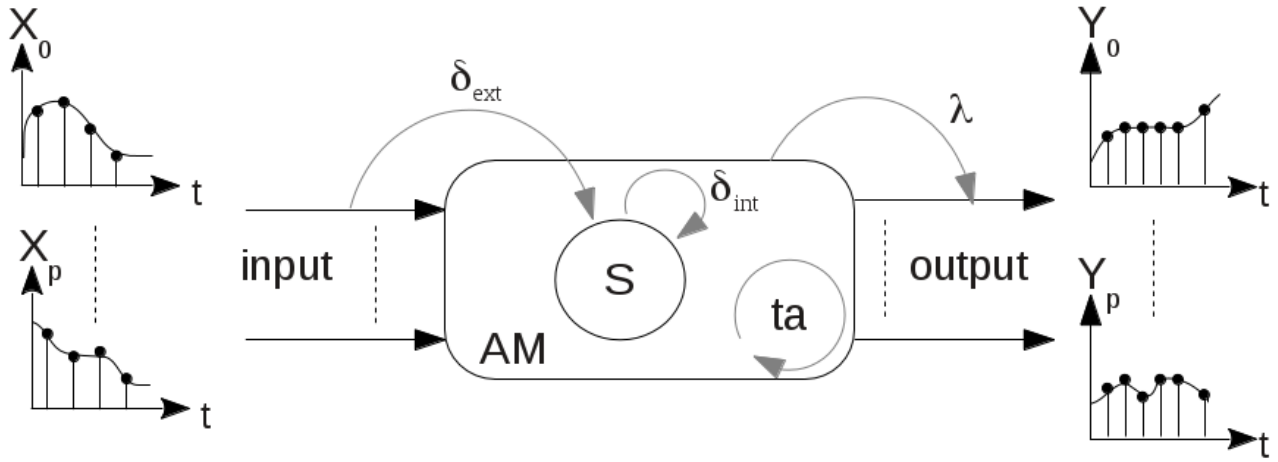
# DEVS Attributes

- DEVS formalizes what a model is, what it must contain, and what it doesn't contain
  - for example, experimentation and simulation control parameters are not contained in the model
- DEVS represents a System of Interest (SoI) with well-defined boundaries.
  - This is critical because composing legacy M&S requires respecting such boundaries for the constituent SoIs.
  - Unfortunately, unlike DEVS, much legacy M&S doesn't have clear, unambiguous definitions for the underlying system boundaries.
- DEVS distinguishes the simulator from the model. Moreover, it enables a DEVS-compliant simulator to execute a DEVS-compliant models correctly and efficiently.
- DEVS defines what's necessary to compose modularized models that will be executable by a compliant simulator.
- DEVS separates the simulator's target hardware environment from the model.
- Indeed, a DEVS model can be executed on multiple different simulators,
  - Including those on desktops (for development) and those on high-performance platforms, such as multi-core processors.

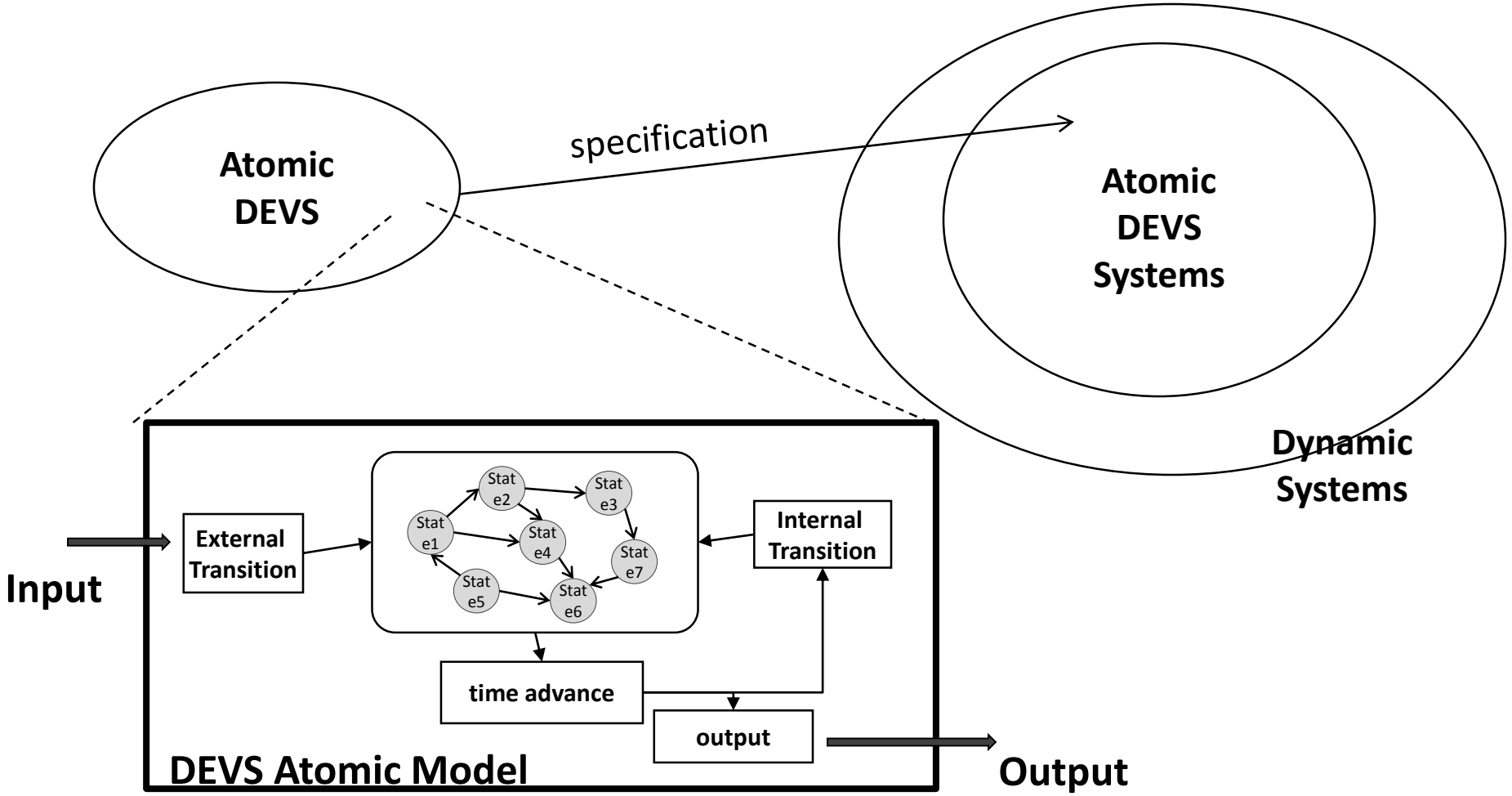
# DEVS Basic Structure (Atomic Model)

$M_{pDevs} = < X_M, Y_M, S, \delta_{int}, \delta_{ext}, \delta_{con}, \lambda, ta >$, where

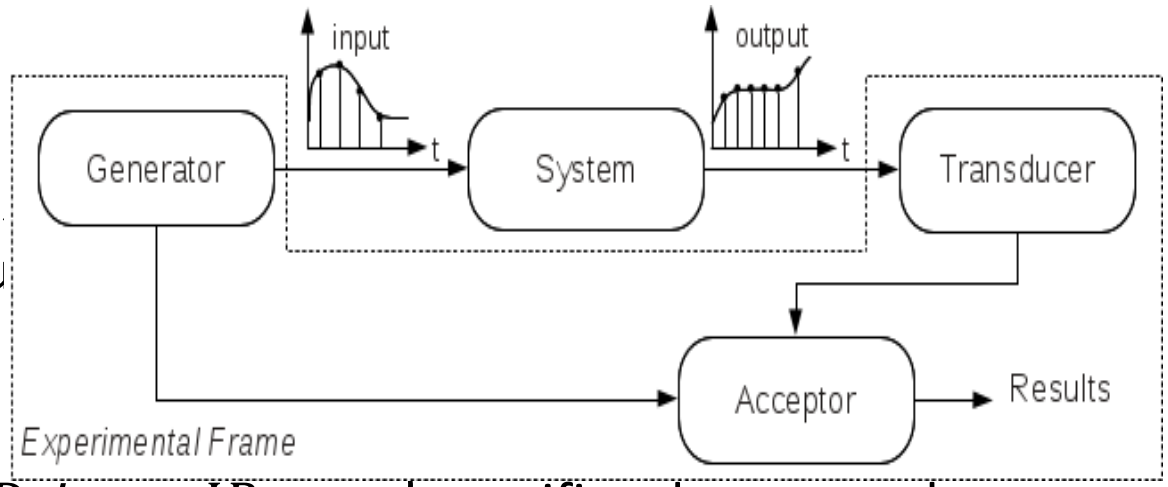
- *X is the set of input values,*
- *S is the set of sequential states,*
- *Y is the set of output values,*
- δint is the internal transition function dictating state transitions due to internal events,
- δext the external transition function dictating state transitions due to external input events,
- λ is the output function generating external events at the output,
- *ta is the time-advance function which allows to associate a life time to a given state.*



# DEVS Atomic Models as System Specifications



# DEVS Coupled Model is




$N = \langle X, Y, D, \{M_d | d \in D\}, EIC, EOC, IC$

where *X, Y* = sets of input and output port-valu

*D* = namespace of DEVS component models

$M_d$ = a DEVS component model

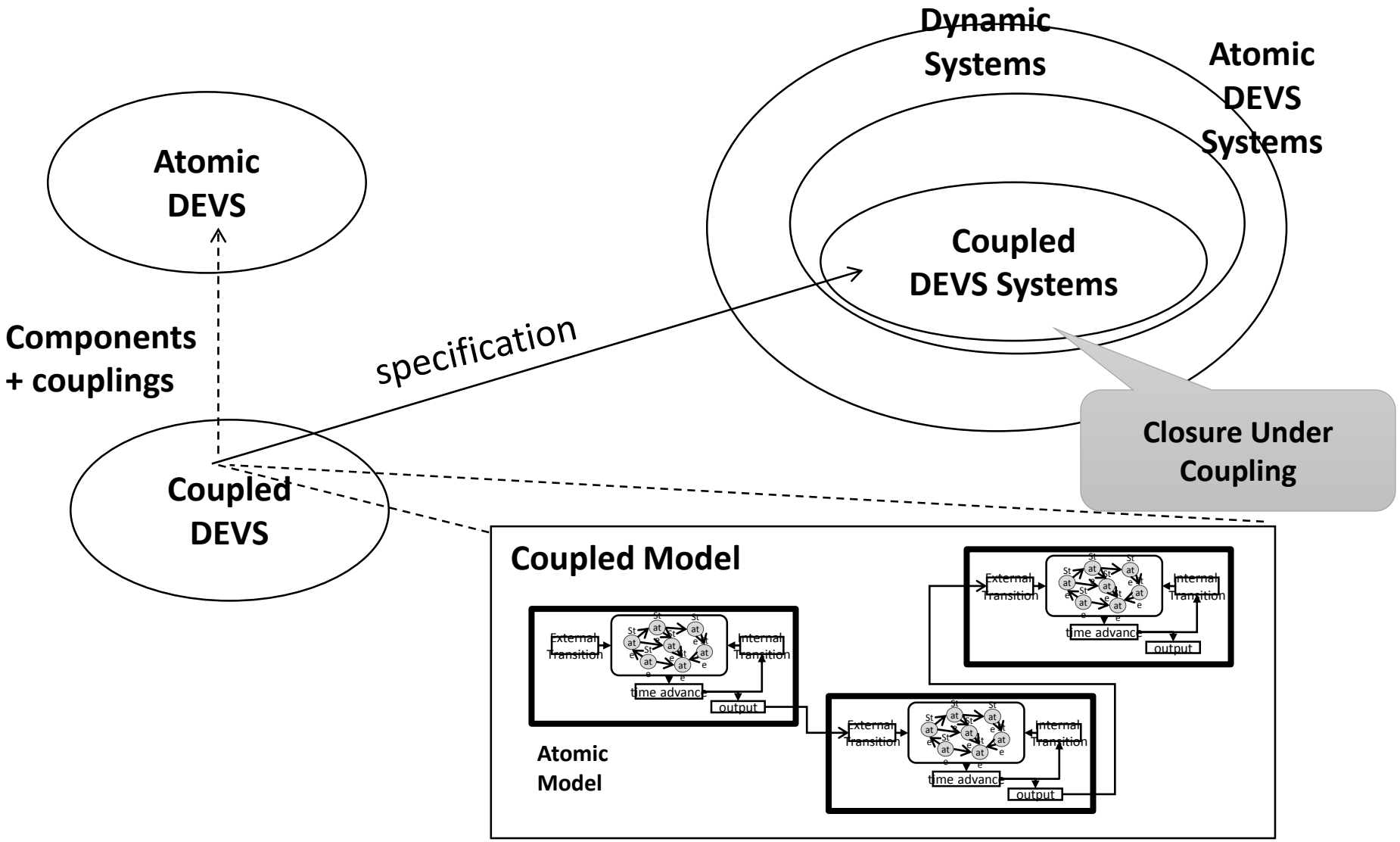
$EIC \subseteq \{((N, ip_N), (d, ip_d)) | ip_N \in IPorts, d \in D, ip_d \in IPorts_d\}$ specifies the external input coupling connecting external inputs of *N* to component model inputs

$EOC \subseteq \{((d, op_d), (N, op_N)) | op_N \in OPorts, d \in D, op_d \in OPorts_d\}$ specifies the external output coupling connecting external outputs of *N* to component model outputs

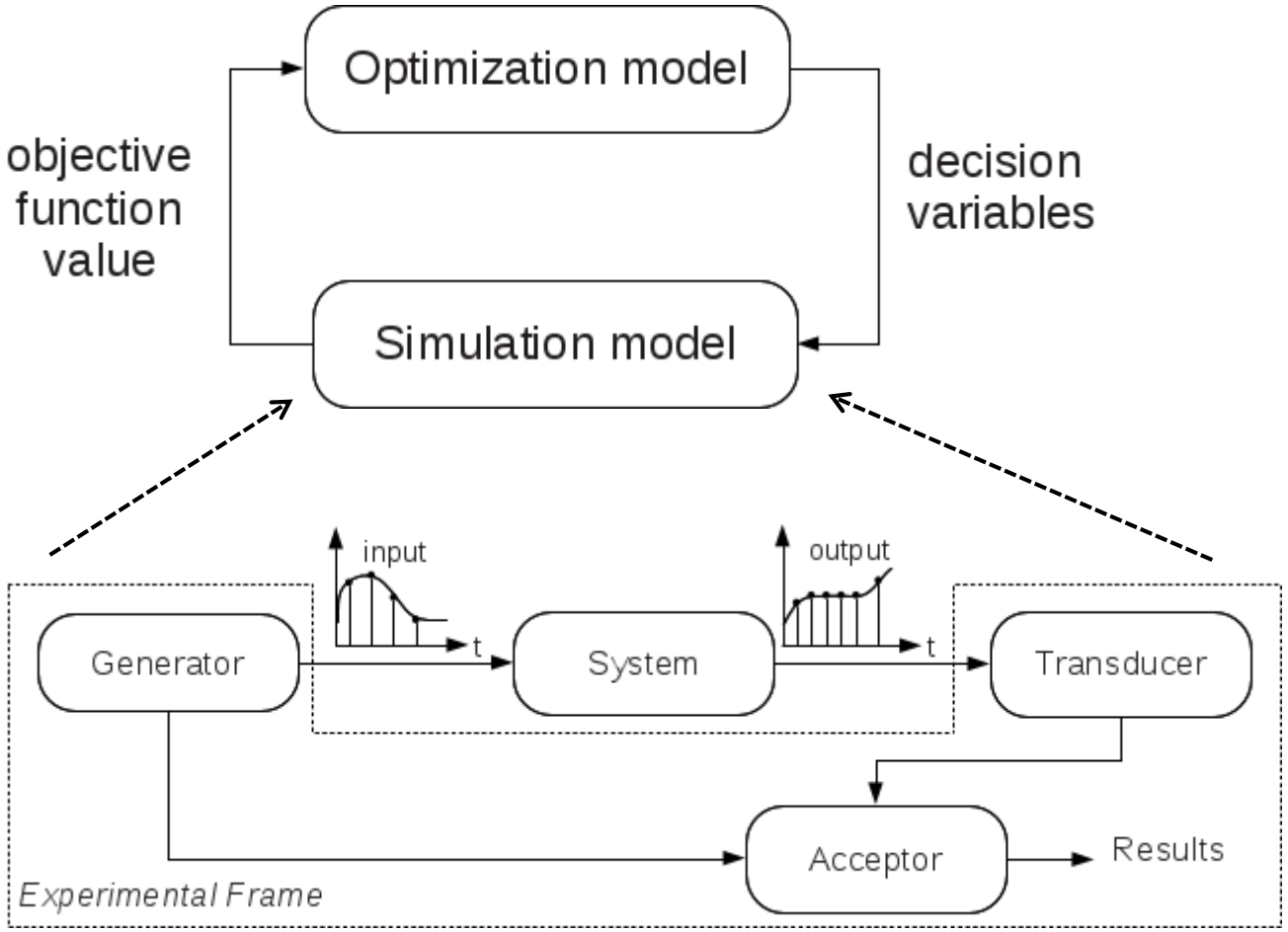
$IC \subseteq \{((a, op_a), (b, op_b)) | a, b \in D, op_a \in OPorts_a, ip_b \in IPorts_b\}$ specifies the internal coupling among component model inputs and outputs

*Select* is a tie-breaking function resolving state transitions in the case of event collisions

# DEVS Coupled Models as System Specifications

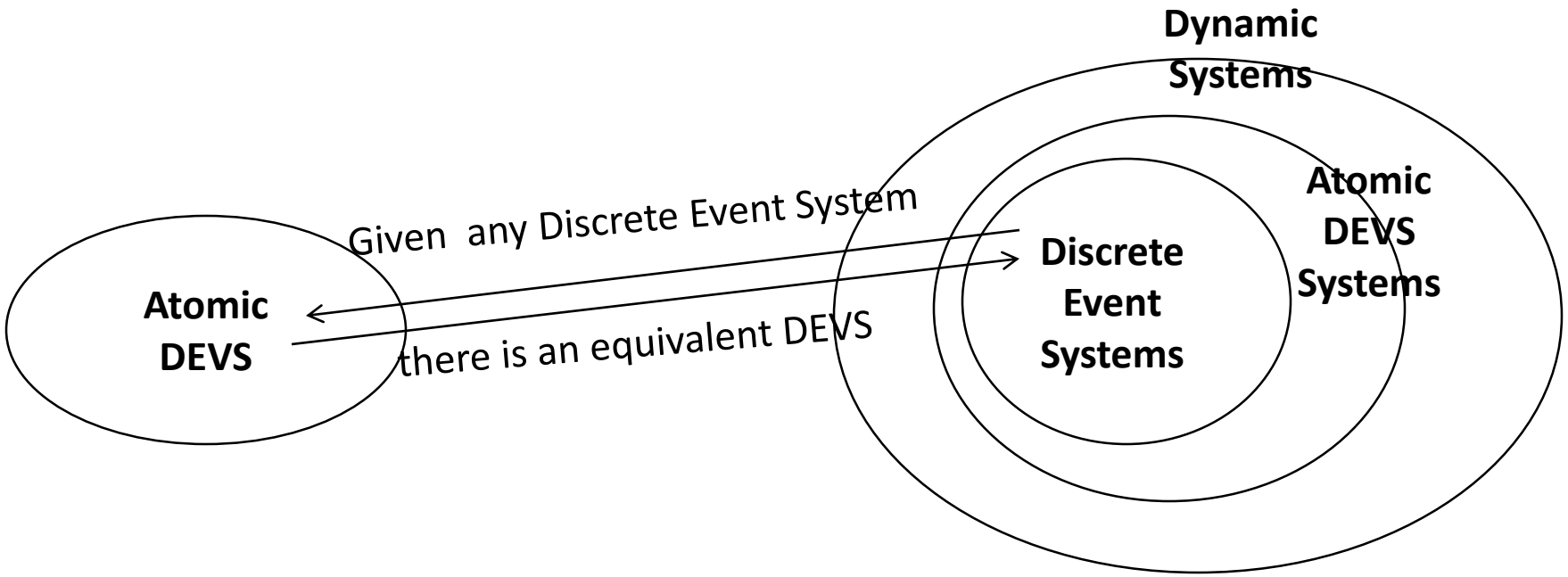


# DEVS Hierarchical Composition: Optimization via Simulation *



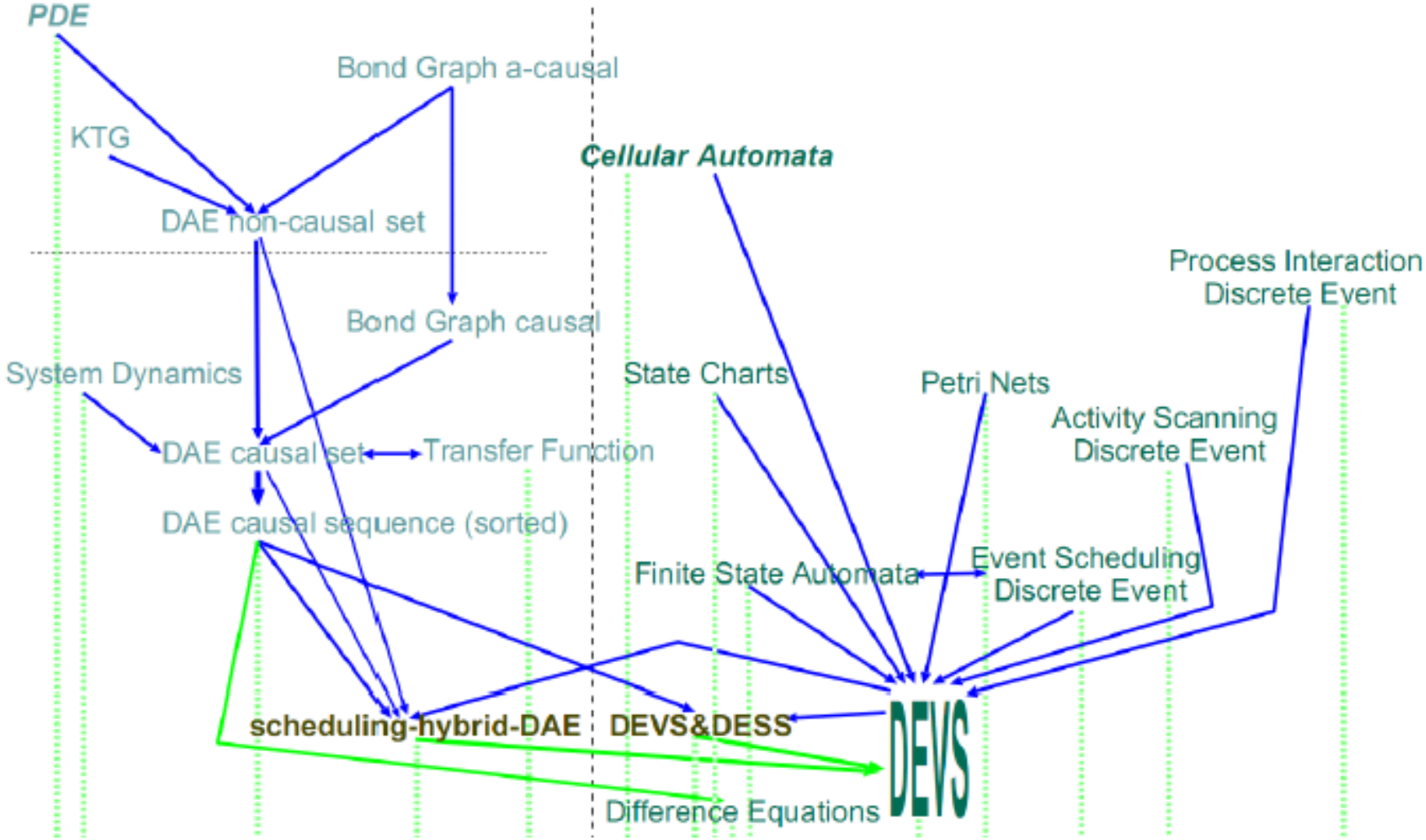

* Optimization via Simulation of Catchment Basin Managment using Discrete-Event Approach
J.F. Santuccia,1,, L. Capocchia,1

# DEVS Universality and Uniqueness for Discrete Event Systems



# DEVS as Universal "Assembly Language"
# Formalism Transformation Graph

[Vangheluwe 2000]



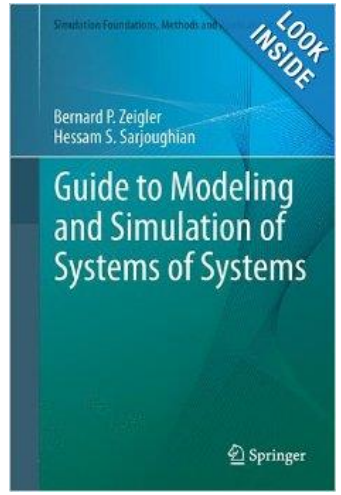
# DEVS M&S Development Environments

| | | |
|---|---|---|
| ADEVS | ADEVS is a C++ library for developing discrete event simulations based on the Parallel DEVS and DSDEVS formalisms. | USA |
| CD++ | CD++ is a general toolkit written in C++ which allows the definition of DEVS and Cell-DEVS models. | Canada |
| DEVS/HLA | HLA-compliant DEVS environment | USA |
| DEVSJAVA | environment for developing DEVS-based models. | USA |
| DEVSim++ | DEVSim++ is an environment for Object-Oriented Modeling of Discrete Event Systems. | Korea |
| GALATEA | multi-agent systems to be simulated in a DEVS multi-agent platform | Venezuela |
| GK-DEVS | DEVS formalism that models and simulates 3-Dimensional Multi-component system. | Korea |
| JAMES II | JAva-based Multipurpose Environment for Simulation II (JAMES II) supports for many formalisms\ | Germany |
| JDEVS | JDEVS enables discrete-event general purpose object-oriented component based | Corsica, France |
| LSIS DME 0.1.0 | DEVS simulator with graphical interface. | Marseilles, |
| SimStudio | DEVS-based PlatForm for the Specification of Simulation Systems | Blaise Pascal, France |
| PowerDEVS | general purpose software tool for DEVS modeling and simulation of hybrid systems. | Argentina |
| Python DEVS | ATOM3 is a tool for multi-paradigm modeling | Canada |
| SimBeans | Component-based software development and Simulation development by modular hierarchical composition of components that adhere to a framework | Austria |
| SmallDEVS | SmallDEVS is an experimental DEVS-based simulation package for Squeak Smalltalk. | Czech Republic |
| VLE | VLE (Virtual Laboratory Environment) is a multimodelling platform based on several DEVS extensions. VLE provides a complete C++ API for DEVS based simulation and a GUI for the graphical specification of the strucutre of the model the definition of experimental frames and the visulalization of results. | Toulouse, France |

# DEVS Applications

- **The French National Institute for Agricultural Research (INRA) chose VLE to integrate its stock of existing agriculture models and to develop new simulation capabilities using DEVS. The DEVS/VLE combination was preferred among many commercial and academic contenders**
- **Intel Corp, Semiconductor Supply-Chain Manufacturing Systems, Optimization/Scheduling,**
- **Usinor, Steel Production and Control**
- **Defense Information Systems Agency, USA – automated testing of net-centric information systems**
- **Military Systems, Korea - Various applications including training and systems engineering**
- **Ballistic Missile Agency, USA – Reference Architecture for myriad missile use cases**
- **US Air Force, Cognitive Systems Research integration of conventional models**
- **National Health Care System Model Development, USA**

# Vision: an IDE specifically tailored to a full DEVS development environment

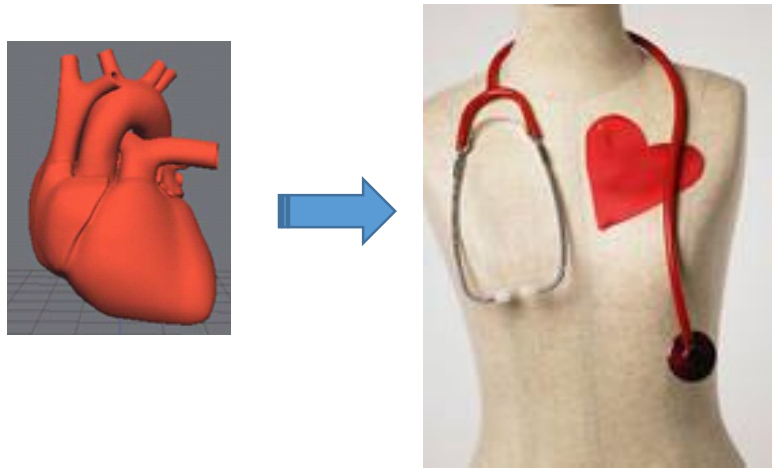



- **MS4 Modeling Environment (Me) offers a complete set of tools to support all functions needed to create DEVS models and simulate them within, and *externally* to, the environment.**

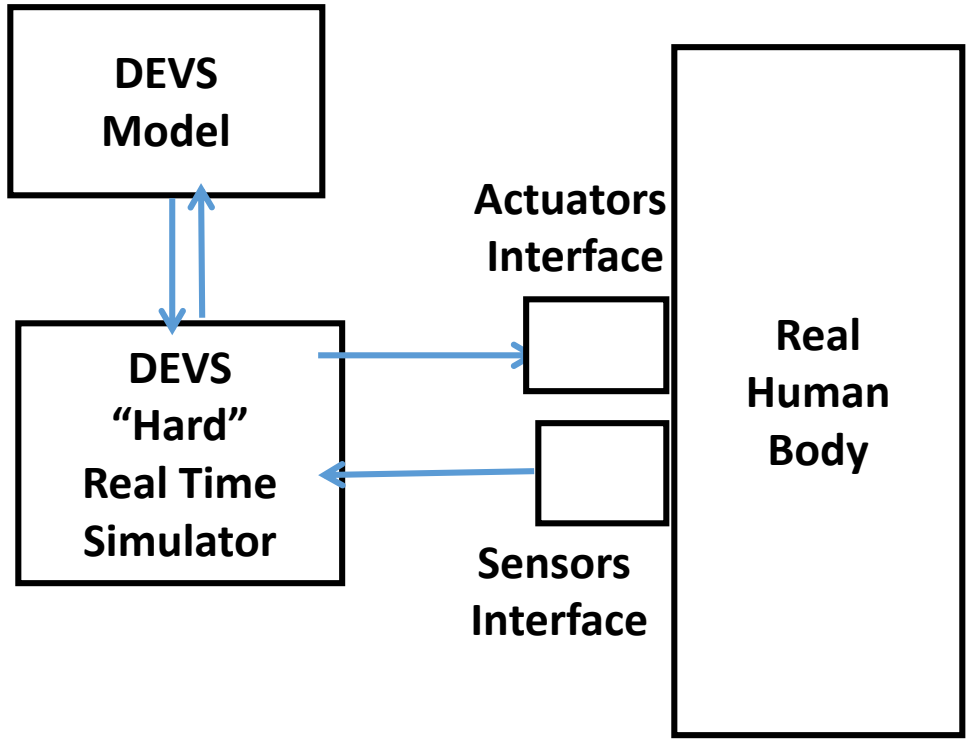
- **MS4 Me employs Xtext and EBNF grammar within the Eclipse Modeling Framework on the Rich Client Platform, and the Graphical Modeling Project**
  - **Eclipse open source community supports of extensible language development and other programming frameworks.**
  - **Xtext framework provides a set of languages and tools to create a programming language and automatically generate its implementation in the Java Virtual Environment.**

# Part 2 DEVS Meeting the Challenges

# **Fishwick's Challenge:** How to replace Real Component by Simulation?

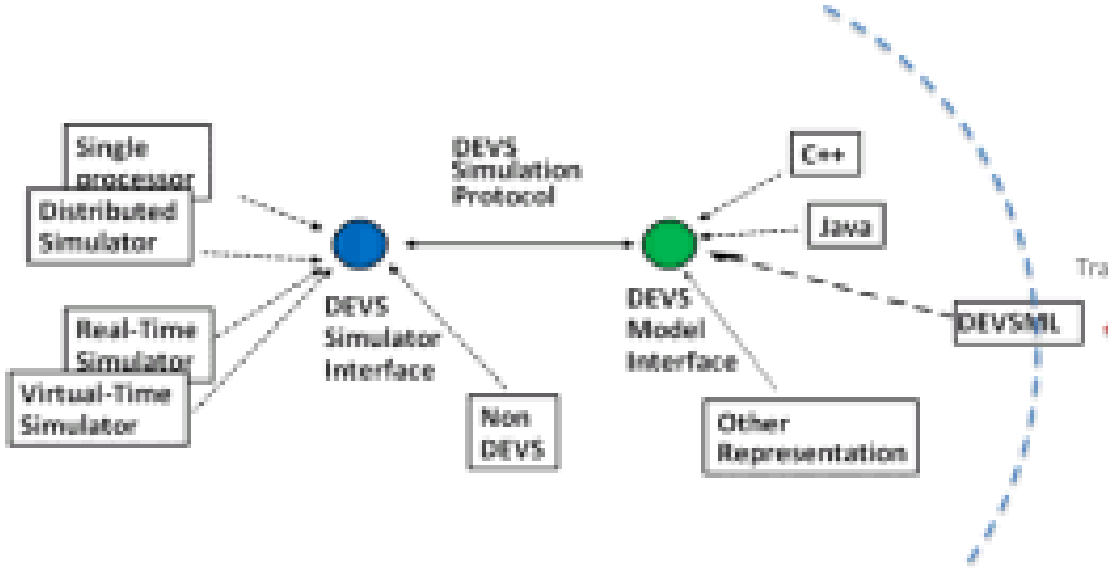


- **Futuristic 3D Printed Heart**
- **Replace real heart**
- **Must be indistinguishable from real heart**
- **Must operate in identical dynamic time frame**



# DEVS Simulation Protocol

## *Both Virtual and Real Time*




- general simulation protocol that prescribes specific mechanisms for:
  - declaring who takes part in the simulation (component model simulators = federates)
  - declaring how federates exchange data
  - executing an iterative cycle that
    - controls how time advances (time management)
    - determines when federates exchange messages (data exchange management)
    - determines when federates do internal state updating (state update management)

• The protocol guarantees correct simulation in the sense that, if the federates are DEVS models then the federation is also a well-defined DEVS coupled model.

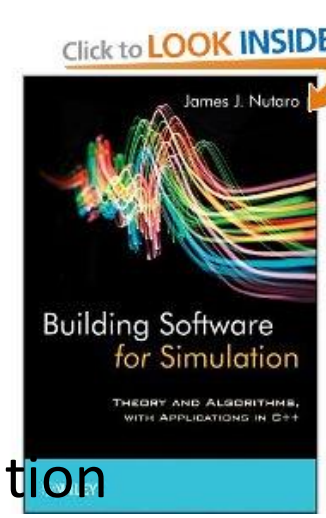
• Distinct from HLA, the DEVS protocol prescribes specific *time advance, data exchange, and state update* management processes.

# Reference Implementation of the DEVS Protocol

Represents in principle how all implementations work. Specifies the following:

- Each component model is assigned to a (distinct) Simulator (federate)
- The federation (coupled model) is assigned to a Coordinator
- The Coordinator and Simulators interact to realize the DEVS simulation protocol, i.e.,
  - the coordinator controls the execution of the ***next event***, invoking the prescribed DEVS internal and external event simulator operations
  - The reference implementation is not intended to be efficient
  - it provides a reference point an unbounded set of possible equivalent implementations that can have desired performance properties
  - E.g. Decentralize coordinator role for distributed simulation
  - E.g. Change to system clock for real-time simulation

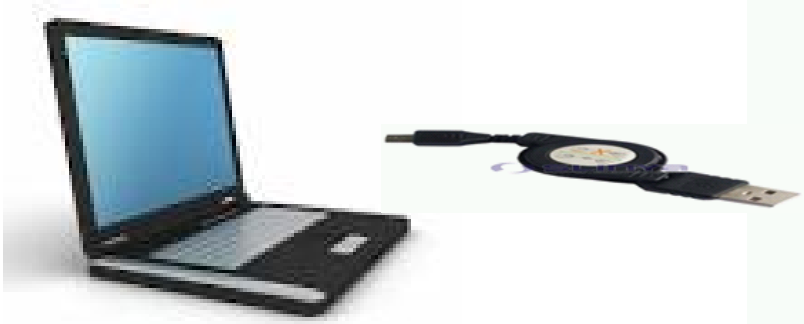
# Example DEVS PADS Simulator




- ADEVS is an open source implementation of the DEVS simulators that targets high performance, multi-core and multi-processor computer systems.
- Uses conservative/optimistic simulation algorithms to enable the parallel execution of DEVS models
- These algorithms exactly reproduce the behavior of the DEVS reference simulators
- Distinguishes ADEVS from simulation engines derived from the logical process approach
- ADEVS supports both discrete event and continuous dynamic systems within the DEVS framework
  - Simulates the interaction of sub-systems characterized by discrete event dynamics (e.g., communication networks and command and control systems) and continuous, physical dynamics (e.g., the trajectories of ballistic missiles and their interceptors)

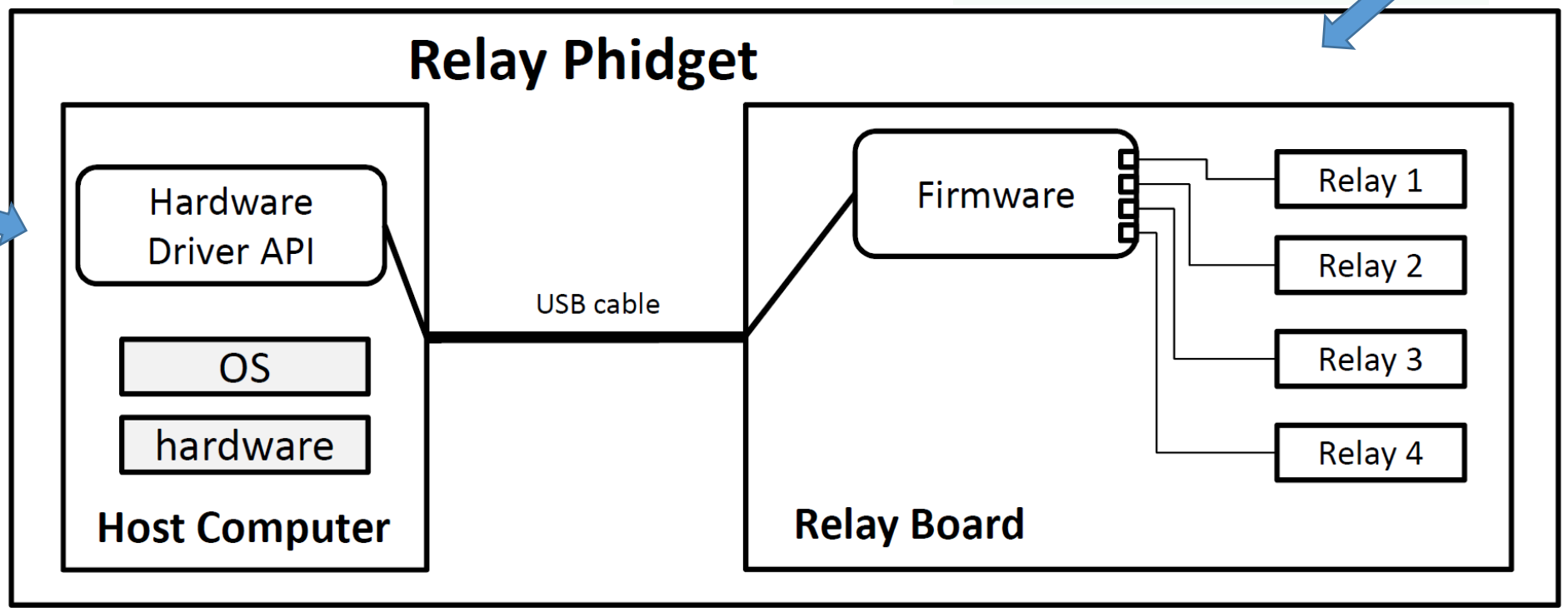
# Real-Time Simulation and Reactive Computational-Physical Systems*

- "Hard Real time" DEVS Simulator
- Runs in same time frame as real system
- DEVS Model Controls system



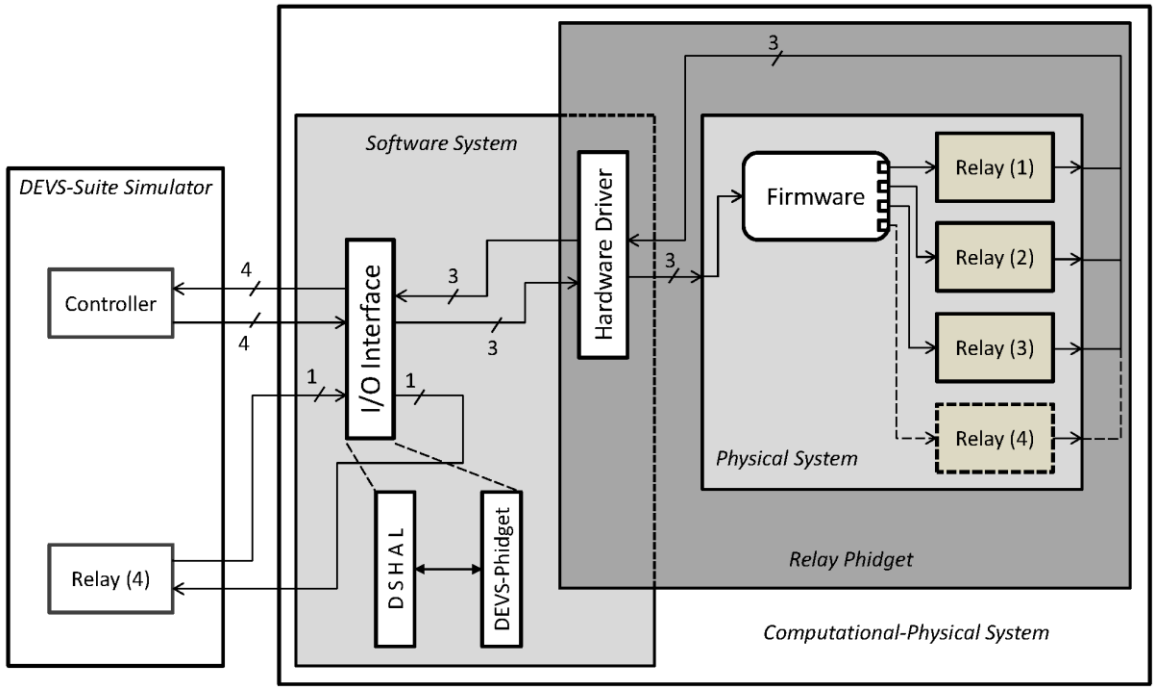
*mechanical relays, firmware, and USB cable = physical part*

***Hardware Driver API = computational part***




Interacting Real-Time Simulation Models and Reactive Computational-Physical Systems Hessam S. Sarjoughian Soroosh Gholamia, Thomas Jackson; Proceedings of the 2013 Winter Simulation Conference

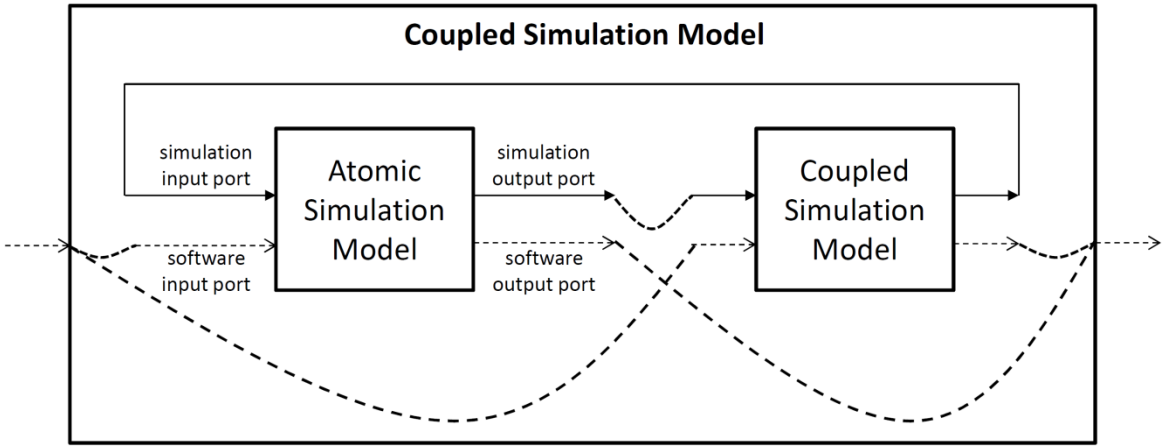
# Separate Modules: Simulator, Software Interface, and Physical System




DEVS Simulation Hardware Abstraction Layer (DSHAL)

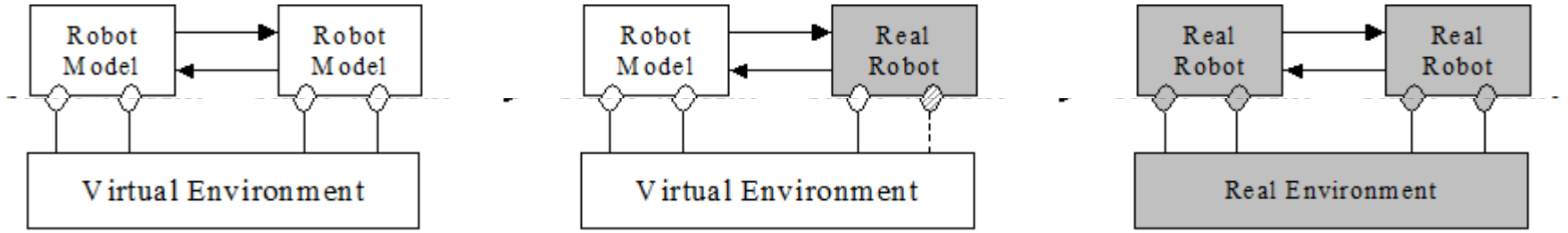
DEVS-Phidget functions listen for input and output events. Operations related to the Hardware Driver include reading/writing to/from memory and attaching/detaching relays

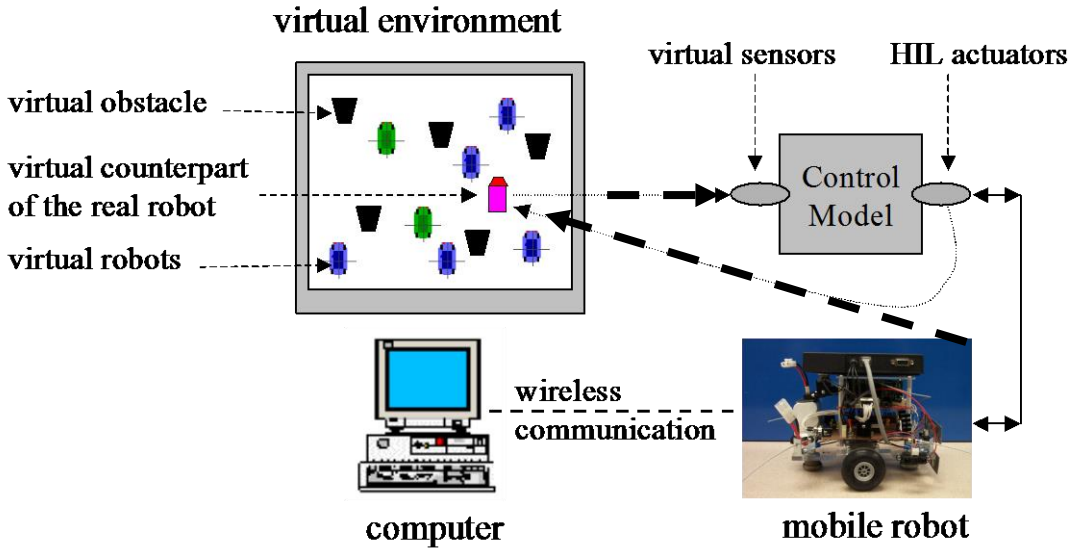
- **DEVS Atomic time advance controlled by real time accurate clock**
- **Define Software output ports (UML 2.0 (OMG 2004)) to send outputs to software system then to hardware**
- **Similarly, receive inputs through software input ports**
- **Allows better control of end-to-end timing**

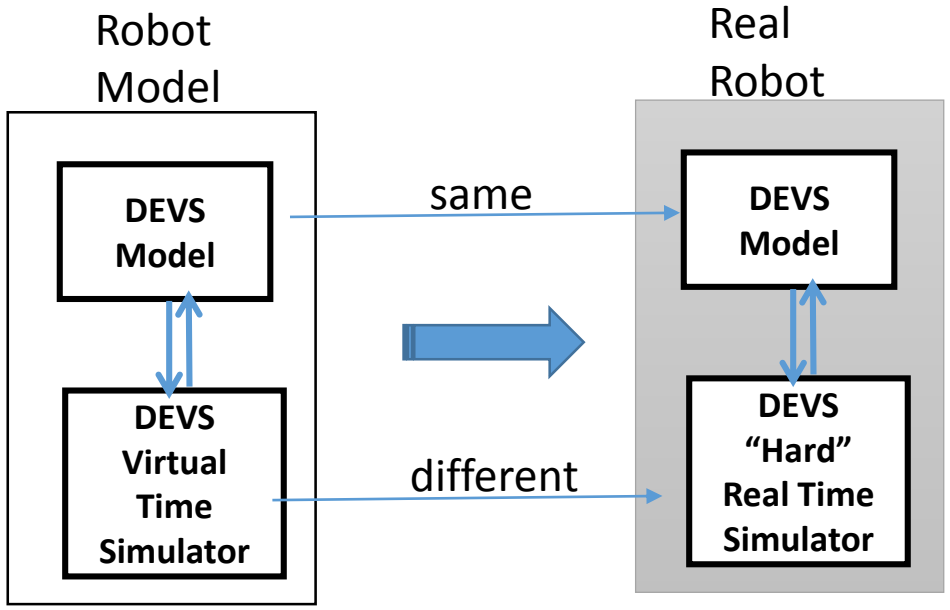


# Model Continuity & Staged System Development:

*Model is same throughout development process*




**A Simulation-Based Virtual Environment to Study Cooperative Robotic Systems**

Xiaolin Hu

# **Uhrmacher's Challenge**: How to test new formalisms and provide user performance information?

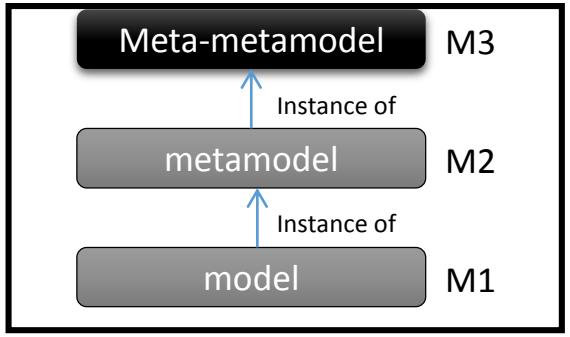
## Meta-Modeling Language Methodology

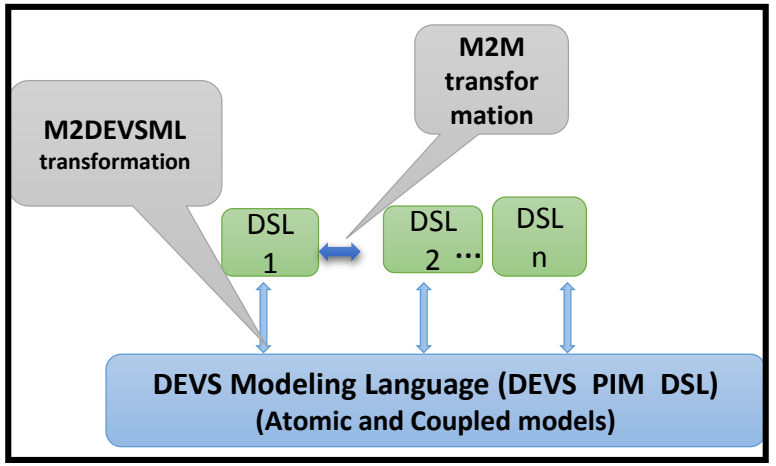
### M1, M2, and M3 Levels

- Domain Specific Languages
  - Defined at M2 Level
  - Oriented to a problem domain/context
  - Meta-modeling process is called Domain Specific Modeling (DSM)
- Key Enabler promoting automated transformations:

### DSL to DSL , DSL to DEVS

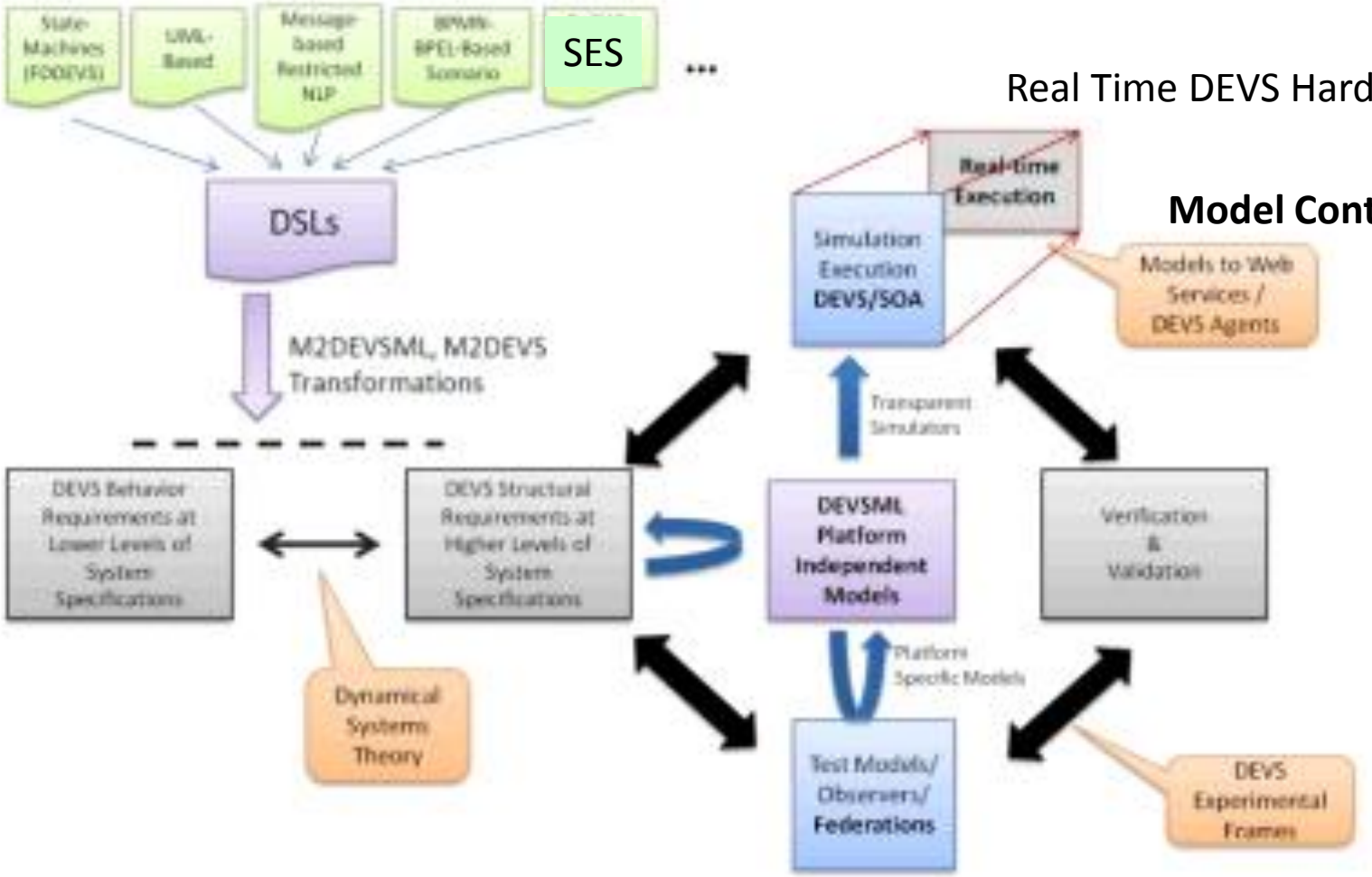
  - Test new formalisms within unified DEVS framework
  - Provide information to user via transformations



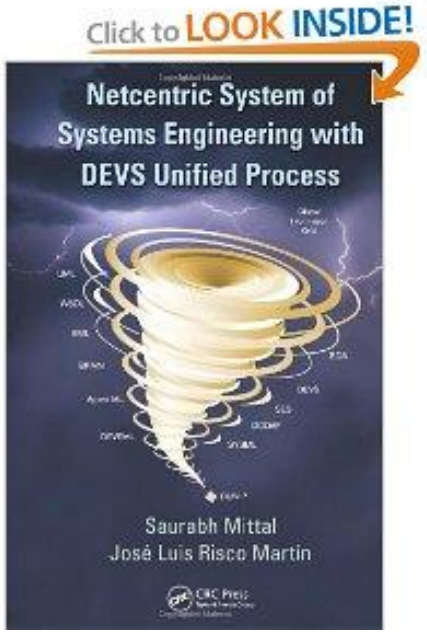
**Wainer's Challenge:** How to interface simulation software with Smartphone APIs?
How to use a more abstract approach to deal with these problems?

## *DEVS Unified Process (DUNIP)*

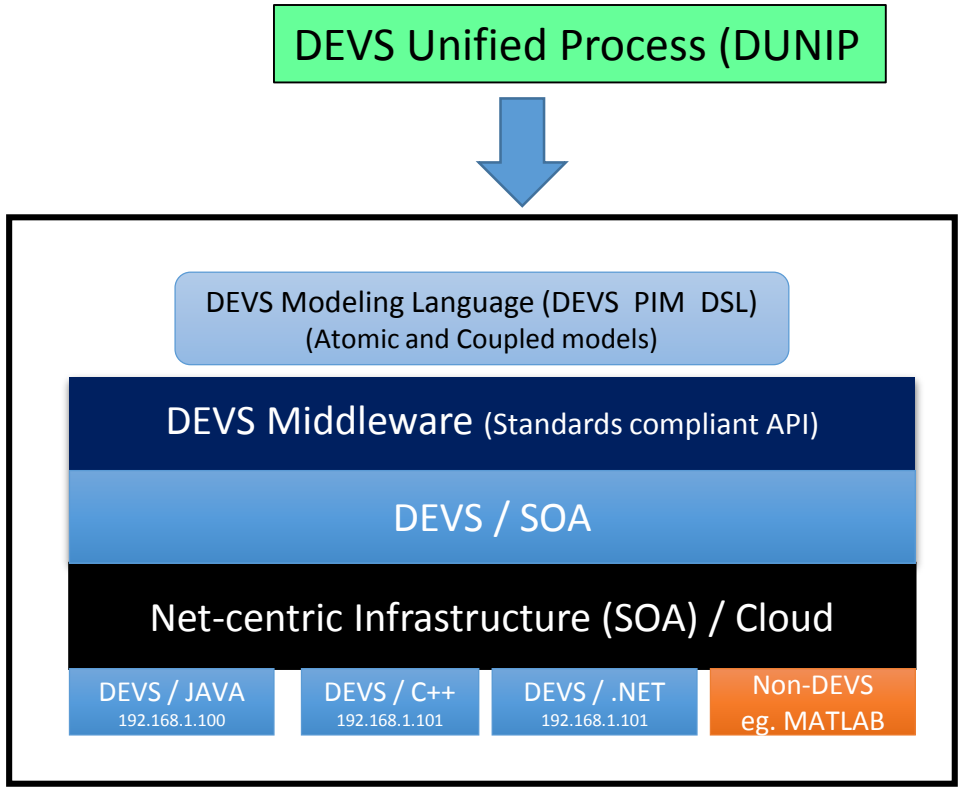



Real Time DEVS Hardware

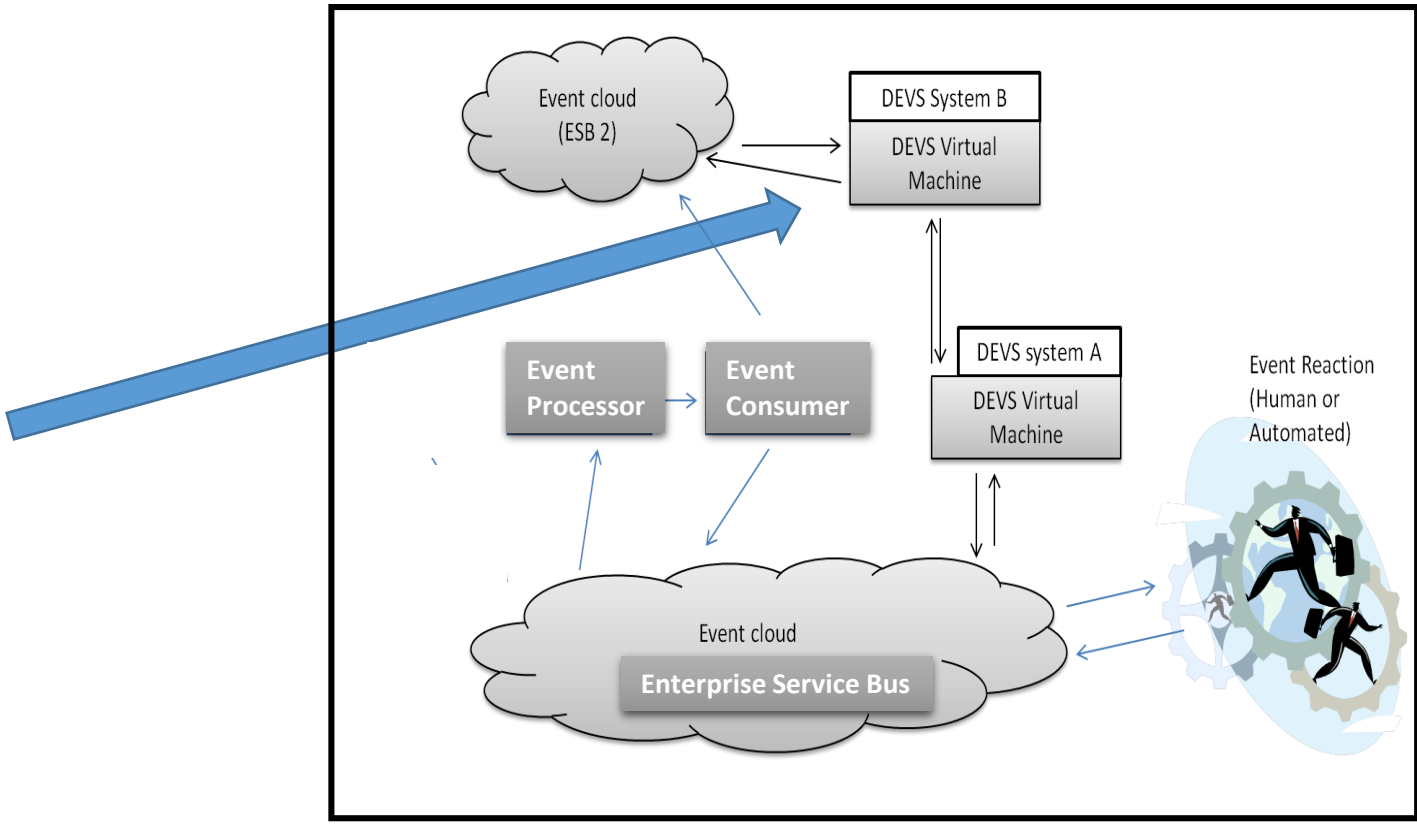
**Model Continuity**



# DEVSML Stack: Netcentric DEVS Virtual Machine and Event Driven Architecture




DEVSML 2.0 stack




Event driven Architecture (EDA)

# DUNIP Compared With Other Model Engineering Approaches

| Features | System/Software Engineering approaches | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | MBE | MBSE | MDSE / DUNIP | MDD4MS | MDE | MDD / MDSD | MDA | MIC |
| Use of DSLs | Y | Y | Y | Y | Y | Y | Y | Y |
| Alignment with Systems theory | Y | Y | Y | - | - | - | - | - |
| DSL representation with metamodeling | - | - | Y | Y | Y | Y | Y | Y |
| Guidance for model transfor-mations | - | - | Y | Y | Y | Y | Y | Y |
| Support for component reusability | Y | Y | Y | Y | - | - | - | Y |
| Code generation/execution | Y | Y | Y | Y | - | Y | Y | Y |
| Code deployment mechanisms | Y | Y | Y | - | - | Y | - | Y |
| Tool support for overall process | - | Y | Y | Y | Y | Y | - | Y |
| Applicable to all domains | Y | Sys. Engg. | Sys. Engg. | Y | Y | Soft. Engg. | Soft. Engg. | Soft. Engg. |

DEVS and Theory of Modeling and Simulation

# Conclusions

- Fishwick's Challenge: **How to replace Real Component by Simulation?**
  - ✓Hard Real-Time DEVS, Model Continuity, Staged System Development
- Uhrmacher's Challenge: **How to test new formalisms and provide user performance information?**
  - ✓DSLs such as UML, SySML, BPMN are analyzable and executable through M2M, M2DEVS and M2DEVSML transformations
- Wainer's Challenge: **How to interface simulation software with Smartphone APIs? How to use a more abstract approach to deal with these problems?**
  - ✓DEVSML supports netcentric DEVS Virtual Machine for fast deployment and transparent simulation framework
  - ✓EDA and DUNIP together brings M&S to complex netcentric environments
  - ✓Roles of DSL designer to design M&S services and DSL end-user to use M&S Services

# Limitations and Future

- DEVS supports model transformations from systems dynamics perspective
  - ➢ But Needs augmentation from semantic and pragmatic perspectives – build on DEVS metamodeling language methodology: e.g., connect ontology with modeling theory
- Future applications:
  - Cloud Manufacturing using simulation
  - M&S using Cloud
  - Health Care