



# Model-Based Design: Challenges and Opportunities

Janos Sztipanovits  
ISIS, Vanderbilt University


Automation to Realize Embedded Systems From High-Level Functional Models  
DATE 2008, Munich, Germany  
March 10, 2008

Sztipanovits

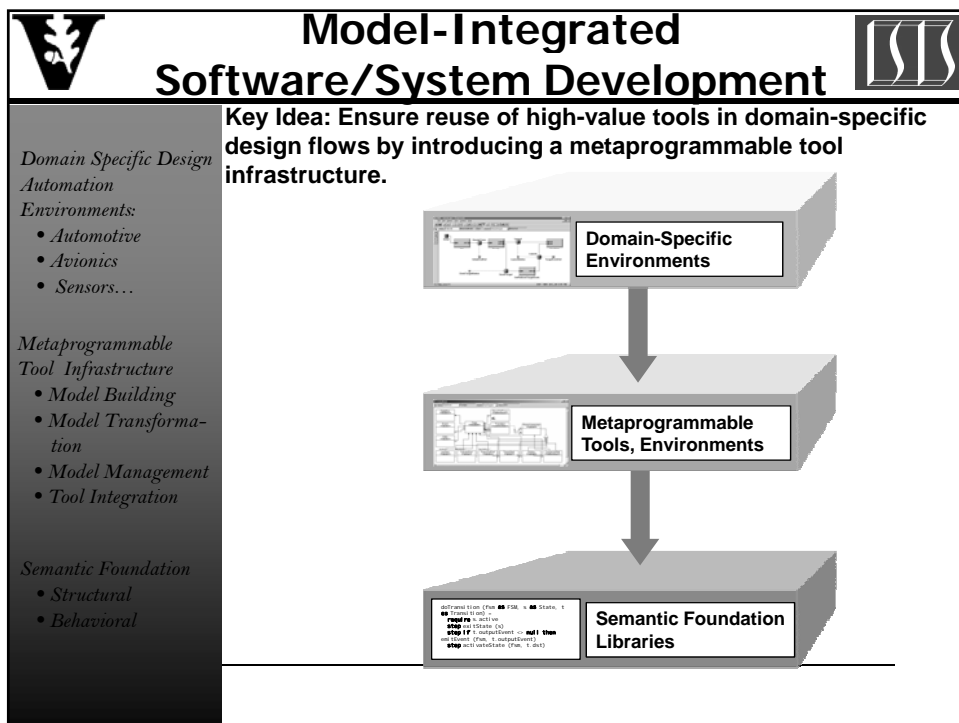
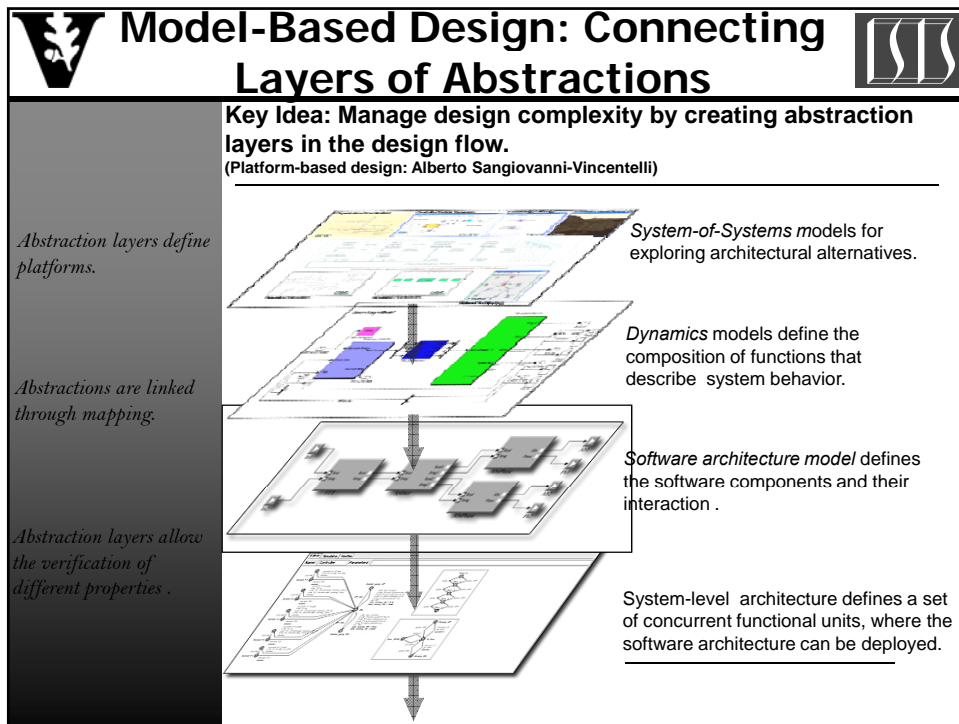


## Overview



- 
- **Introduction**
  - **Architecture exploration**
  - **Separation of Concerns**
  - **Dealing with semantics**
  - **Summary**

Sztipanovits: 2





## Agile Design Automation

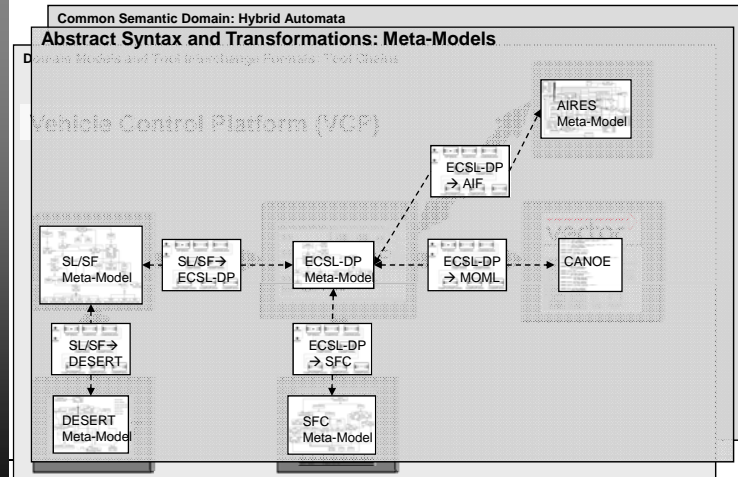


Abstraction layers are defined by formally specified DSML-s.

Metamodels are used for expressing relationship among models used across the design flow.

Metamodels of model transformations describe the "glue" connecting analysis tools to the design flow.

**Key Idea: Integrate domain and tool specific models through metamodeling and model transformations.**



## GME: Metaprogrammable Modeling Tool

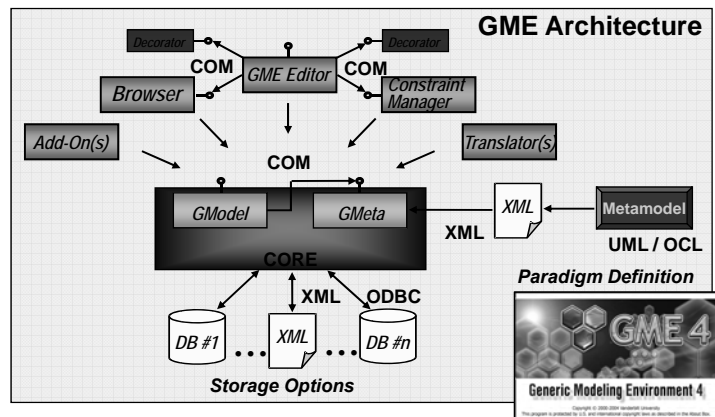


The Generic Modeling Environment (GME) is used for building metamodels and is customized by metamodels (metaprogrammability).

MetaGME combines language constructs for metamodeling and for specifying concrete syntax of DSML-s.

Metacircularity enables changing metalanguages without changing the tool.

The metamodel for MetaGME is the meta-metamodel – defined in MetaGME.



- Configuration through UML and OCL-based metamodels
- Extensible architecture through COM
- Multiple standard backend support (ODBC, XML)
- Multiple language support: C++, VB, Python, Java, C#



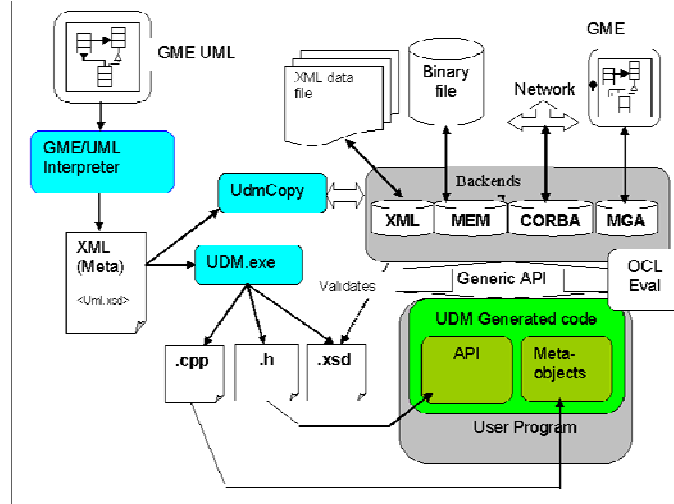
## Model Data Management: The UDM Tool Suite



The goal of UDM is to have a conceptual view of data/metadata that is independent of the storage format.

UDM provides uniform access to data/metadata such that storage formats can be changed seamlessly at either design time or run time

UDM generates a metadata/paradigm specific API to access a particular class of data.



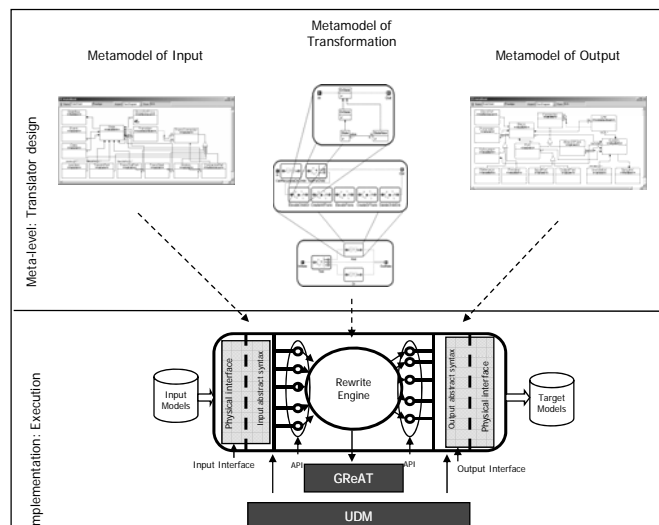
## Model Transformation: The "Workhorse" of MIC



MIC Model transformation technology is based on graph transformation semantics

Model transformations are specified using metamodels and the code is automatically generated from the models.

Models of transformations are expressed in a DSML and built in GME.



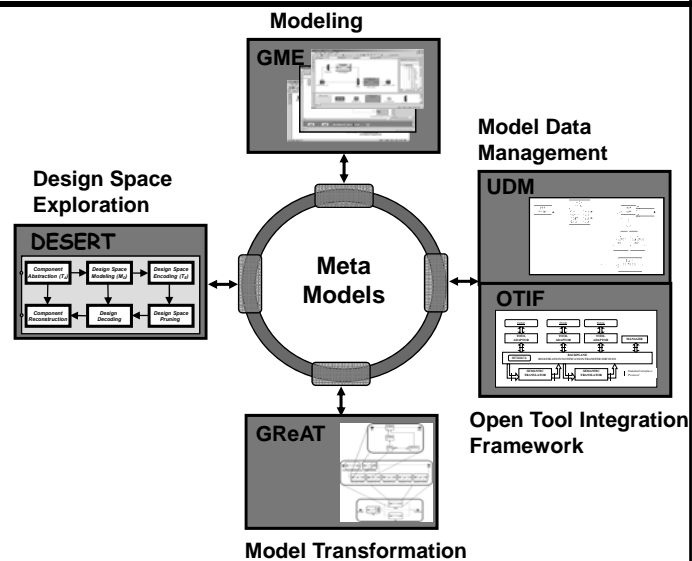


## Integrated MIC Tool Suite



*The MIC tool suite is metaprogrammable: each component can be customized to domain specific modeling languages by means of metamodels.*

*All tools are available from a quality controlled open source repository.*



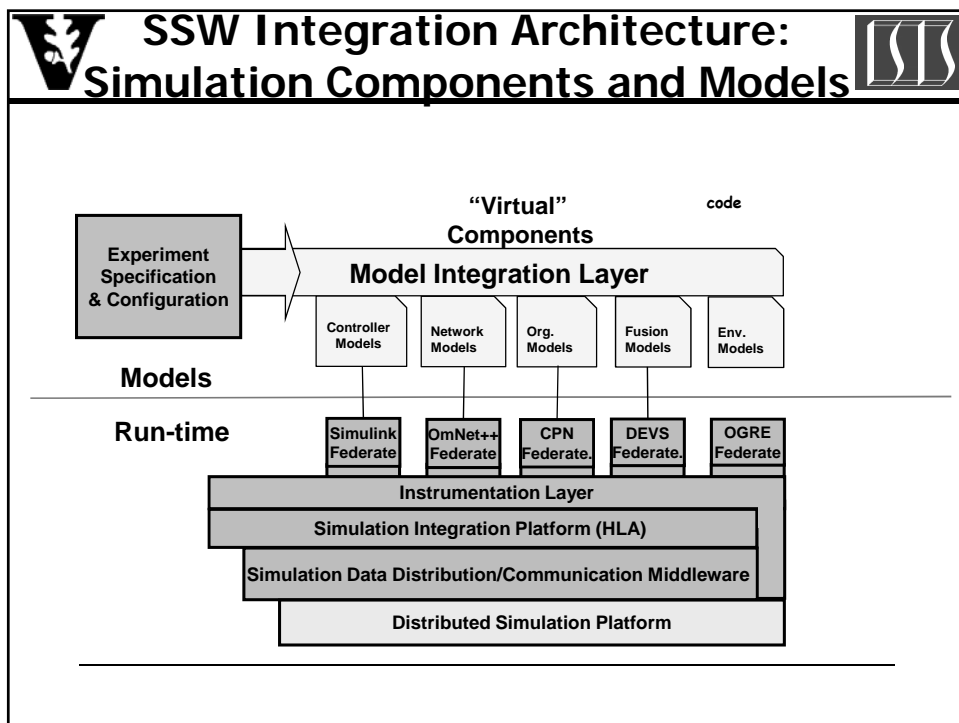
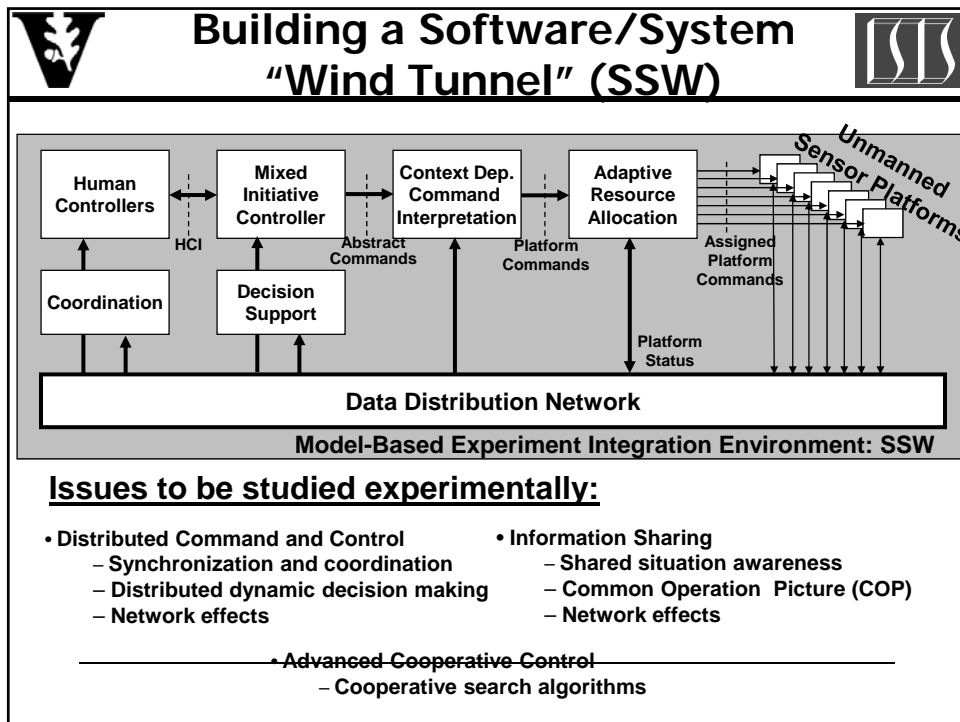
ESCHER Quality Controlled Repository:  
<http://escher.isis.vanderbilt.edu>



## Overview

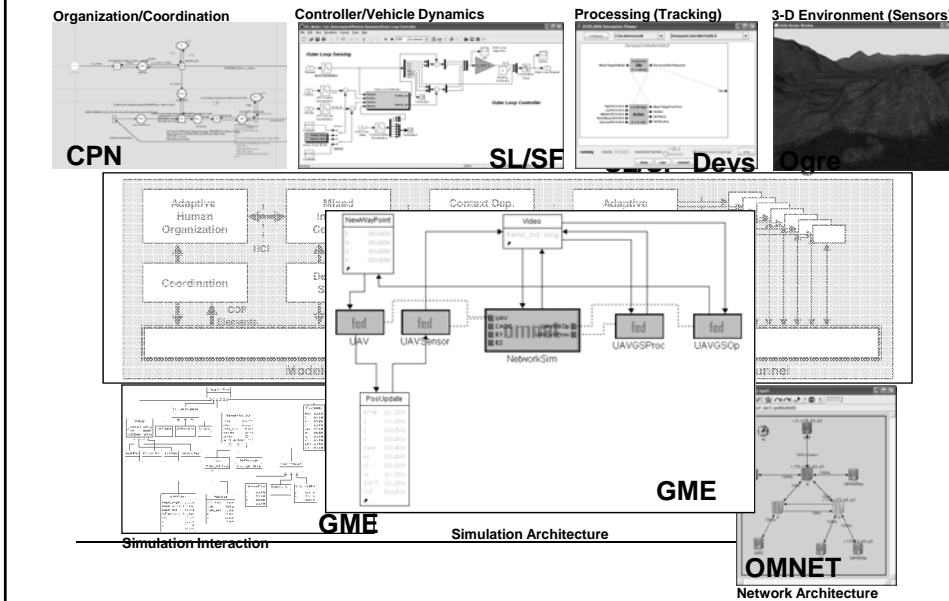


- Introduction
- ▪ Architecture exploration
- Separation of Concerns
- Dealing with semantics
- Summary





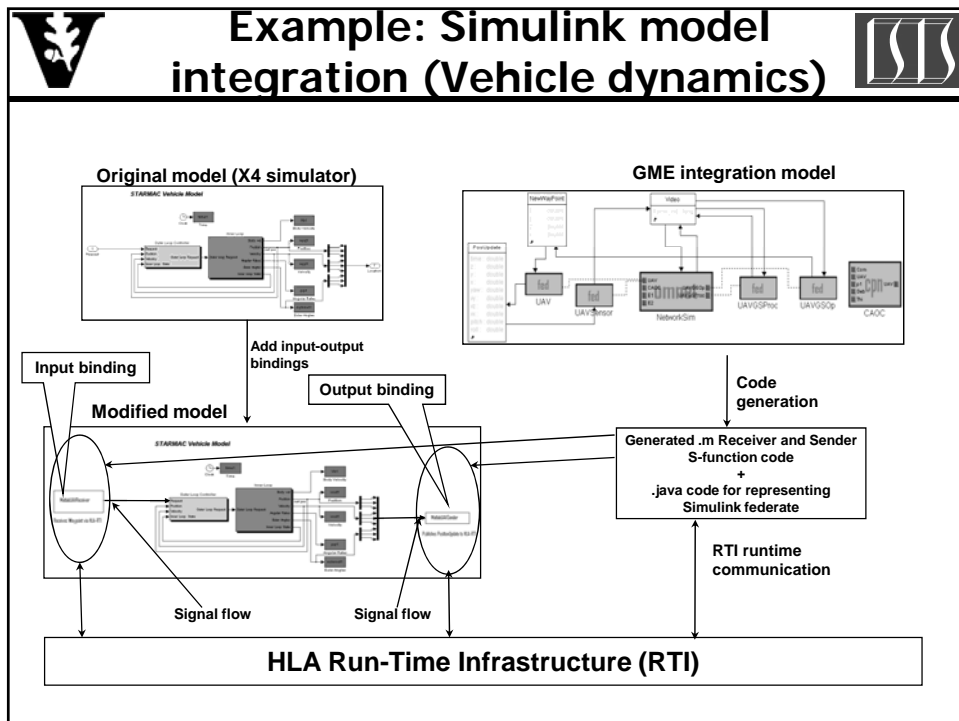
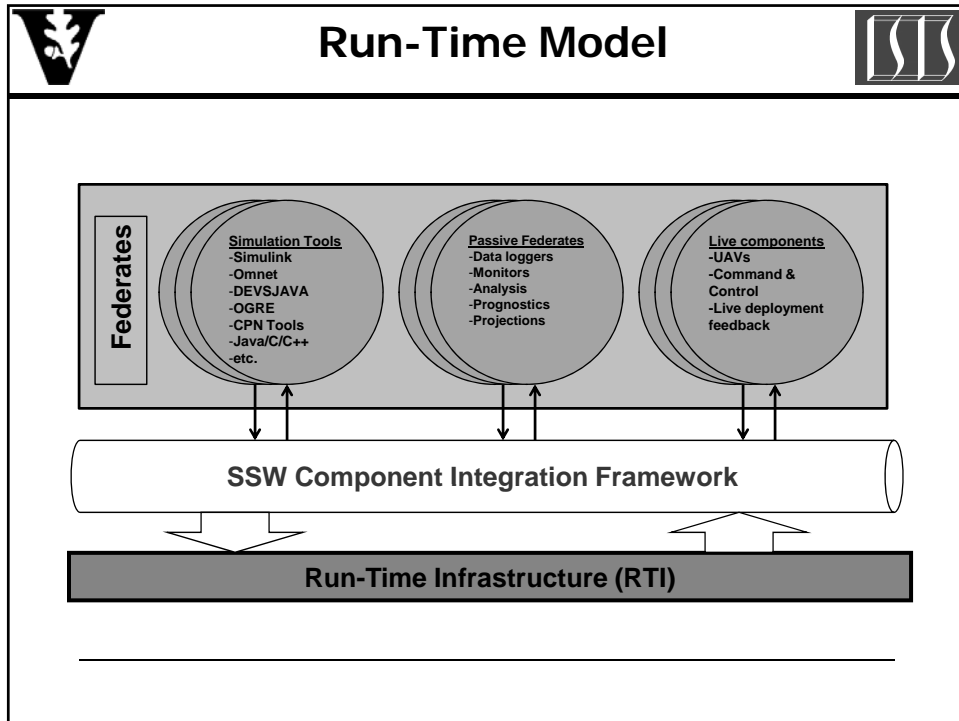
## Model Integration



## Component Integration: The High Level Architecture (HLA)

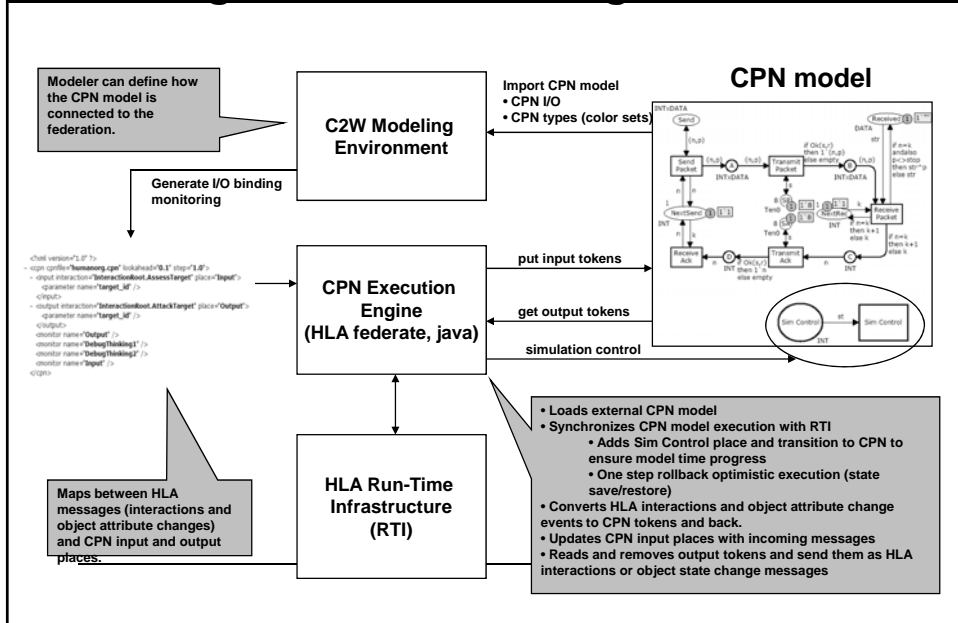


- An IEEE standard for “interoperable” and “reusable” models and simulations.
  - Most used specification (also used in the demo) is IEEE HLA 1.3 (1998)
  - Most recent specification is IEEE HLA 1516 (2000+)
- Primary goal is to provide a general purpose infrastructure for “distributed” simulation and analysis.
- Software implementing the HLA specification is called Run-Time Infrastructure (RTI).
  - Several commercial and open-source RTIs are available.
  - In the demo we used an open-source RTI PORTICO v0.7.1 implemented in Java language (<http://porticoproject.org/>).



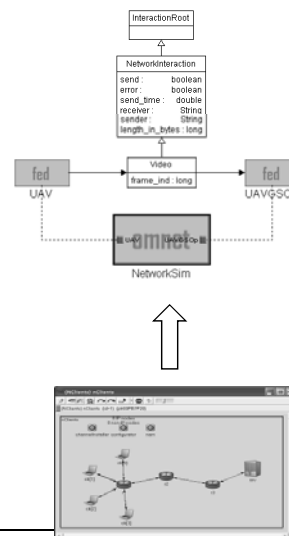


## Example: Colored Petri Net model integration (Human org. models)



## Example: Network simulation integration

- Omnet, Inet packages
  - Omnet is a generic discrete event simulation package (module specification with .ned files, implementation in c++, modular, customizable plugin architecture)
  - Inet: network protocols for omnet (ip, wireless, ad hoc, etc)
- Omnet integration
  - Challenges
    - Scheduler integration
    - Data type mapping
  - SSW network support
    - Build in NetworkSim federate, takes care of omnet scheduler synchronization and data conversion
    - Built in network interaction (NetworkInteractions)
    - Derive interactions from the NetworkInteraction to specify custom data types
    - Derived interactions will be sent through the network simulator
    - Federates can be connected to network endpoints, addressing is based endpoint names





## Challenges and Opportunities in Architecture Exploration



- Test system behavior in its application context before the system is built (SSW concept).
  - Component integration on a discrete event simulation integration platform.
  - Model integration using metamodeling and integration language.
  - Automated, model-based experiment integration.
  - How to deal with dynamic architectures and environments?
- 



## Overview



- Introduction
  - Architecture exploration
  - ▪ Separation of Concerns
  - Dealing with semantics
  - Summary
-

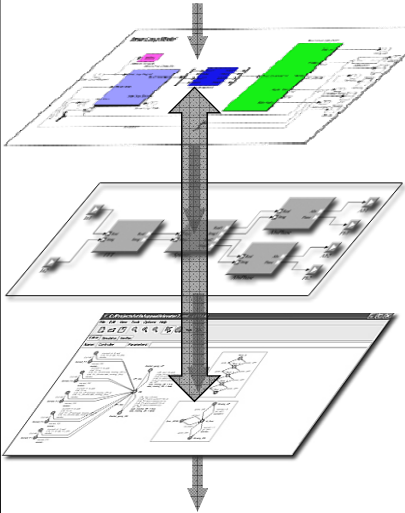
## A Problem With Design Flows: Can we separate design concerns?

*Key properties (such as compositionality!) on abstraction layers depend on assumptions on other layers.*

*Abstractions layers are linked through a web of assumptions.*

*Abstraction layers allow the verification of different properties – but change on any layers may require re-verification.*

**Key Idea: Manage design complexity by creating layers of abstractions in the design flow....but it works only if the layers are decoupled.**

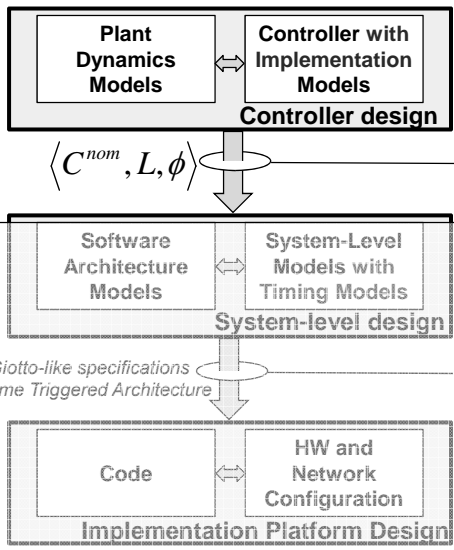


*Controller dynamics* is developed without considering implementation uncertainties (e.g. word length, clock accuracy) optimizing performance.

*Software architecture models* are developed to meet functional specifications, systems engineering constraints, cost constraints (and others).

*System-level architecture* defines implementation platform configuration. If fixed-point arithmetic does not provide required numerical accuracy, time constraints may require modifying platform configuration that will trigger re-verification on all levels.

## Orthogonalization Is Hard, But There is Progress...



*Robust control methods* extended to implementation uncertainties. Result of the design is a *nominal controller*,  $C^{nom}$ , a *Lyapunov Performance Certificate*,  $L$ , (in LMI form) and an implementation complexity measure,  $\Phi$ . (See Boyd and Skaf, Stanford)

**Re-defined specification/implementation interface**

*Software architecture and system-level design* is decoupled (subject to limits) from controller design and includes implementation platform specific abstractions.

**Robust implementation platform designed for decoupling the two design layers**

*Code and HW/Network configuration* is decoupled (subject to limits) from system-level design and allows composition for other properties (reliability).

- Giotto-like specifications  
- Time Triggered Architecture

Sztipanovits: 22

## Component Coefficient Truncation Using Lyapunov Certificate

Control Requirement

Control System Design  
(Specifier)

$\theta^{nom}$ 
 $\mathcal{C}$ 
 $\phi$

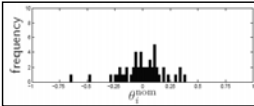
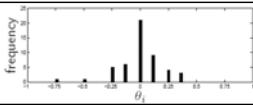
Implementation Design  
(Implementer)

$\theta \in \mathcal{C}$   
 $\phi(\theta)$  is low

- Example control design problem:  

$$x(t+1) = Ax(t) + Bu(t), \quad x(0) = x_0$$

$$u(t) = Kx(t)$$
- The nominal controller  $K^{nom}$  is the LQR optimal feedback controller with double precision floating-point coefficients.
- Admissible controllers  $\mathcal{C}$  are controller that yield an LQR cost that is, at most, 15% suboptimal.
- The complexity measure  $\phi$  is the number of bits required to express  $K$ .
- The best design, which is 14.9% suboptimal, gives only 1.5 bits/coefficients.

Boyd, S., Skaf, J.: "Controller Coefficient Truncation Using Lyapunov Performance Certificate" Proceedings of the European Control Conference, July 2, 2007

Sztipanovits: 23

## Orthogonalization Is Hard, But There is Progress...

Plant Dynamics Models

Controller with Implementation Models

Controller design

$\langle C^{nom}, L, \phi \rangle$

Software Architecture Models

System-Level Models with Timing Models

System-level design

Giotto/TDL-like timing spec. (Henzinger)

PTIDES (Lee)

Time Triggered Architecture (Kopetz)

Code

HW and Network Configuration

Implementation Platform Design

*Robust control methods extended to implementation uncertainties. Result of the design is a nominal controller,  $C^{nom}$ , a Lyapunov Performance Certificate,  $L$ , (in LMI form) and an implementation complexity measure,  $\phi$ . (See Boyd and Skaf, Stanford)*

**Re-defined specification/implementation interface**

*Software architecture and system-level design is decoupled (subject to limits) from controller design and includes implementation platform specific abstractions.*

**Robust implementation platforms designed for decoupling the two design layers**

*Code and HW/Network configuration is decoupled (subject to limits) from system-level design and allows composition for other properties (reliability).*

Sztipanovits: 24



## Challenges and Opportunities in Separation of Concerns



- Orthogonality of design concerns – expressed as abstraction layers (platforms) in design flows - is a common assumption and the primary tool for managing complexity.
- However, orthogonality cannot be achieved by declaration; it needs to be “earned”.
- Achieving robustness in design platforms is a major challenge.

Sztipanovits: 25




## Overview




- Introduction
- Architecture exploration
- Separation of Concerns
- ➔ • Dealing with semantics
- Summary

Sztipanovits: 26



## Specification of DSMLs



$$L = \langle Y, R_Y, C, ([ ]_{i \in I}) \rangle$$

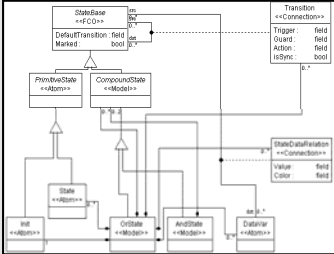
$$D(Y, C) = \{r \in R_Y \mid r \models C\}$$

$$[ ] : R_Y \mapsto R_Y.$$

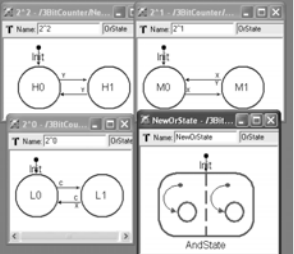
**Key Concept:** Modeling languages define a set of well-formed models and their interpretations. The interpretations are mappings from one domain to another domain.

Y: set of concepts,  
R<sub>Y</sub>: set of possible model realizations  
C: set of constraints over R<sub>Y</sub>  
D(Y,C): domain of well-formed models  
[ ]: interpretations

**Abstract syntax** defines the set of well-formed models using metamodels and metamodeling languages:




MetaGME metamodel of simple statecharts




Model-editor generated from metamodel

Sztipanovits: 27



## Metamodeling is Accepted in Practice – So What Is Wrong?

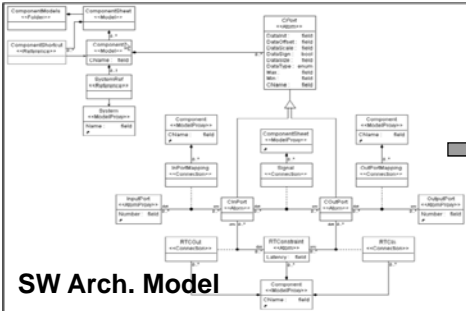


**SW Architecture**  
(DSML<sub>SL/SF,CM</sub>)

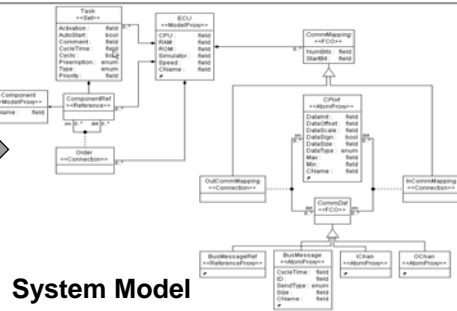
**System Model**  
(DSML<sub>SM,HWA</sub>)

**SW Deployment Model**  
(DSML<sub>SL/SF,CM,SM,HWA</sub>)

**Metamodels** are used to describe the relationship among controller models and SW architecture models. This relationship is expressed as an integration metamodel. Metamodel composition is well established and supported.



**SW Arch. Model**



**System Model**

**SW Deployment: SW Components – System Mapping**

Sz

**We do not realize that metamodels define the structural semantics for DSMLs**



## Metamodeling and Structural Semantics



$$L = \langle Y, R_Y, C, ([ ]_{i \in I}) \rangle$$

$$D(Y, C) = \{r \in R_Y \mid r \models C\}$$

$$[ ]: R_Y \mapsto R_Y.$$

$Y$ : set of concepts,  
 $R_Y$ : set of possible  
 model realizations  
 $C$ : set of constraints  
 over  $R_Y$   
 $D(Y, C)$ : domain of well-  
 formed models  
 $[ ]$ : interpretations

Jackson & Sztiapanovits  
 - EMSOFT 2006  
 - MODELS 2007

Sztiapanovits: 29

**Key Concept:** Structural semantics provides mathematical formalism for interpreting models as well-formed structures.

**Structural Semantics** defines modeling domains using a mathematical structure. This mathematical structure is the semantic domain of metamodeling languages.

**Requirements for the selected formalism:**

- Represent modeling domains (even if infinite).
- Compute if a model is element of the domain.
- Construct well-formed models.
- Provide semantic domain for model transformations.

**Arguments for investigating structural semantics:**

- It is NOT “static semantics” – the structure can be dynamic (e.g. in networked embedded systems)
- Structure-behavior paradigm matches engineering view of modeling
- Model transformations are the workhorse of MIC/MBE.



## Behavioral Semantics



- Given a DSML


$$L = \langle Y, R_Y, C, ([ ]_{i \in I}) \rangle$$

$$D(Y, C) = \{r \in R_Y \mid r \models C\}$$


$$[ ]: R_Y \mapsto R_Y.$$

- Behavioral semantics will be defined by specifying the transformation between the DSML and a modeling language with behavioral semantics.

Sztiapanovits: 30




## Problems with the State of Practice




1. Only “Static Semantics” is defined (informally) with metamodels.
2. Behavioral semantics is implicitly defined by
  - (proprietary) simulation code
  - (proprietary) code generator
  - a user community
3. Behavioral semantics is defined by examples and informal expression of intent.
4. Formal specification developed for some established modeling languages using various formal frameworks with huge effort.
5. ....

No established “standard” methodology and formalism

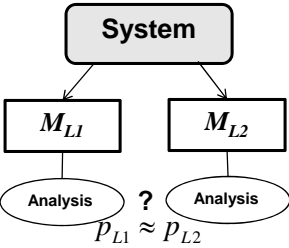
Sztipanovits: 31



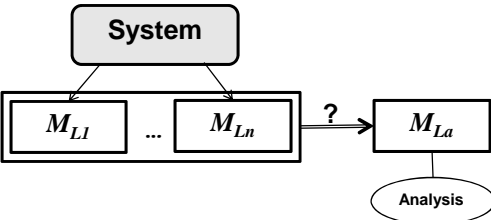
## Why Do We Need It?



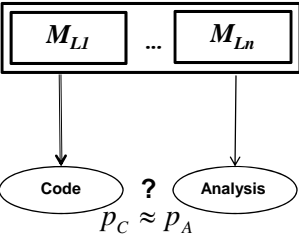
a) Analysis Consistency



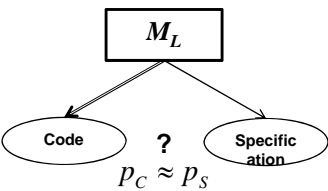
a) Tool Reusability



c) Model-based Code Verification

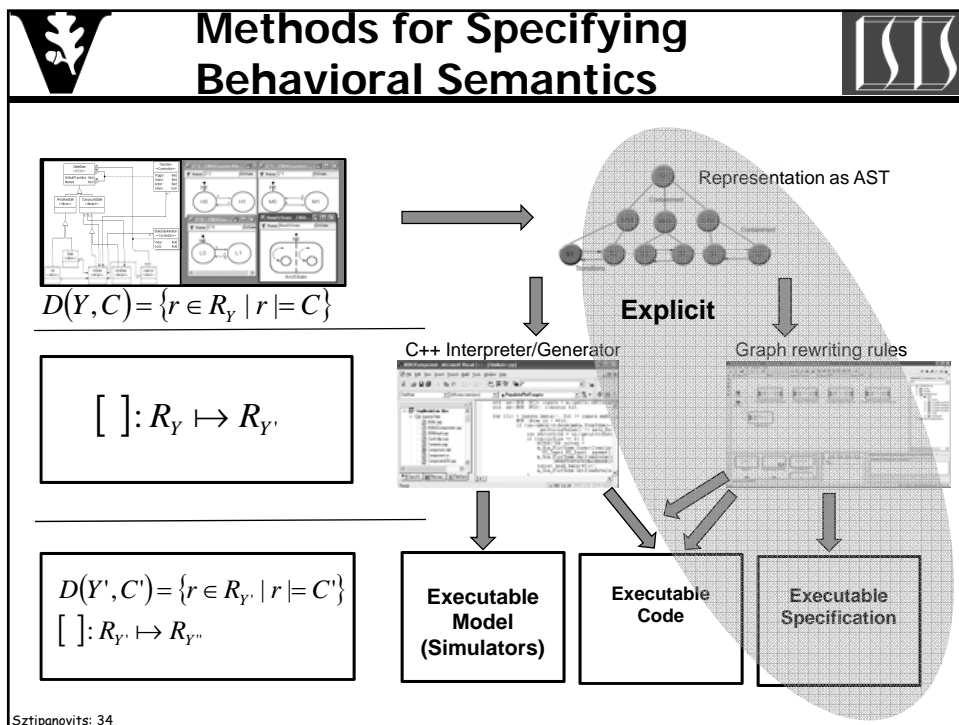
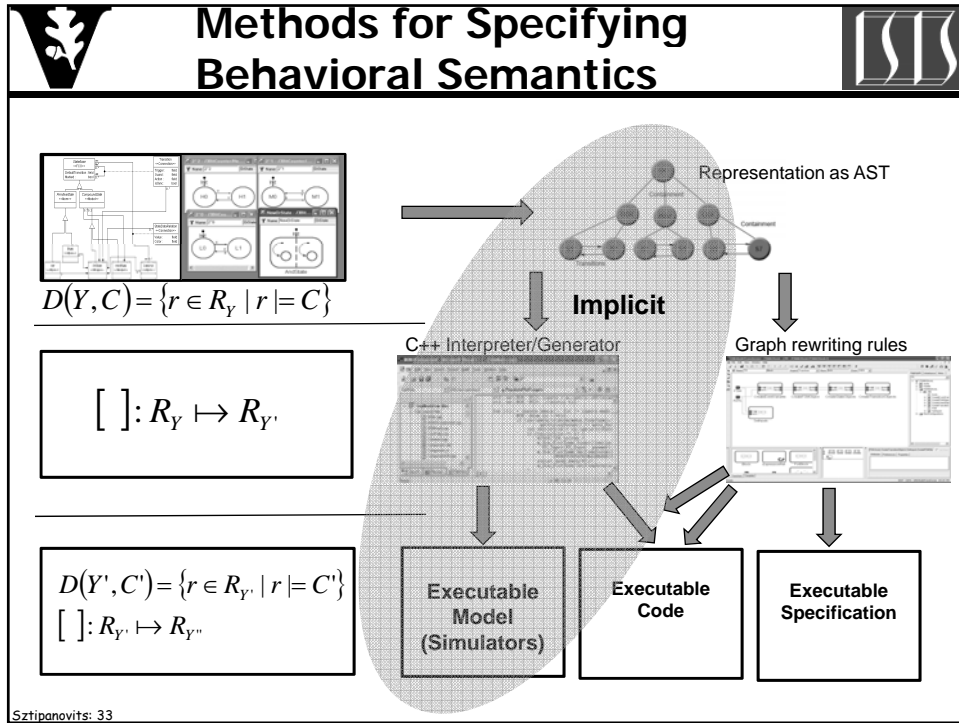


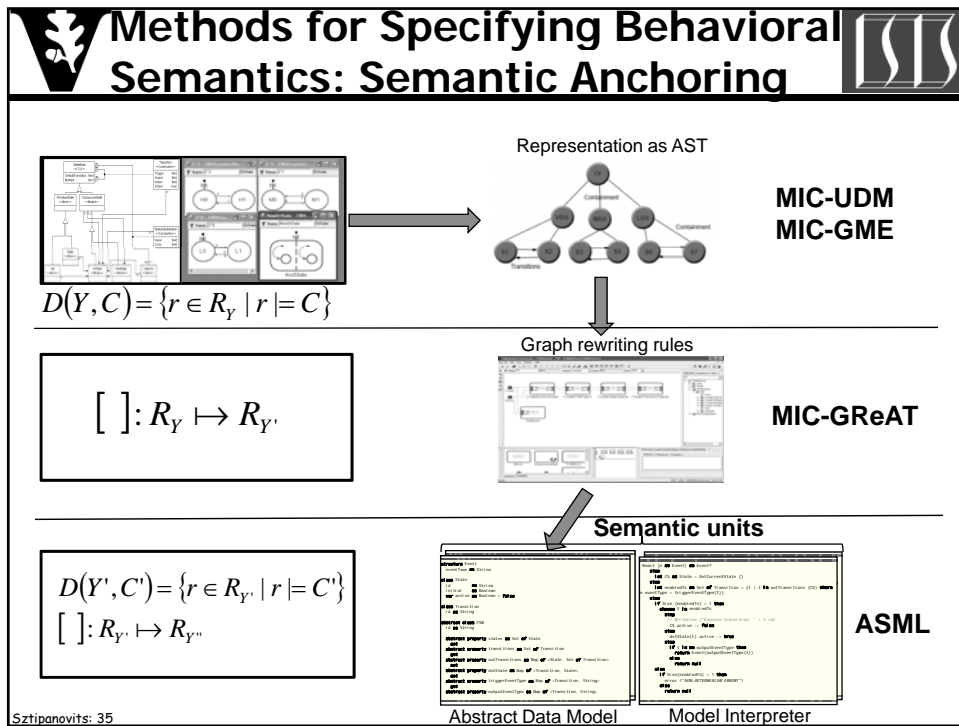
d) Semantic equivalence



Sztipanovits: 32







## Challenges and Opportunities in Defining Semantics of DSMLs

- **Structural semantics** is the foundation for extending model-based design to structurally dynamic systems – a common system category in networked embedded systems.
- **Transformational specification** of behavioral semantics is feasible and model-based design offers direct help in facilitating it.
- **Modeling semantics** is a wicked problem: you need a lot of experience in model-based design just to accept that it may make sense to do it...

Sztipanovits: 36



## Summary



- **Model-based design methods span the full design flow from high-level architecture exploration to code generation and to system integration.**
- **Enables to take another look at the design flow: it becomes also design objective facilitated by agile design automation platforms.**
- **Enables addressing fundamental issues in design technology such as orthogonality and semantics.**

Sztipanovits: 37