Design Space Construction and Exploration

A Model-Integrated Computing Approach

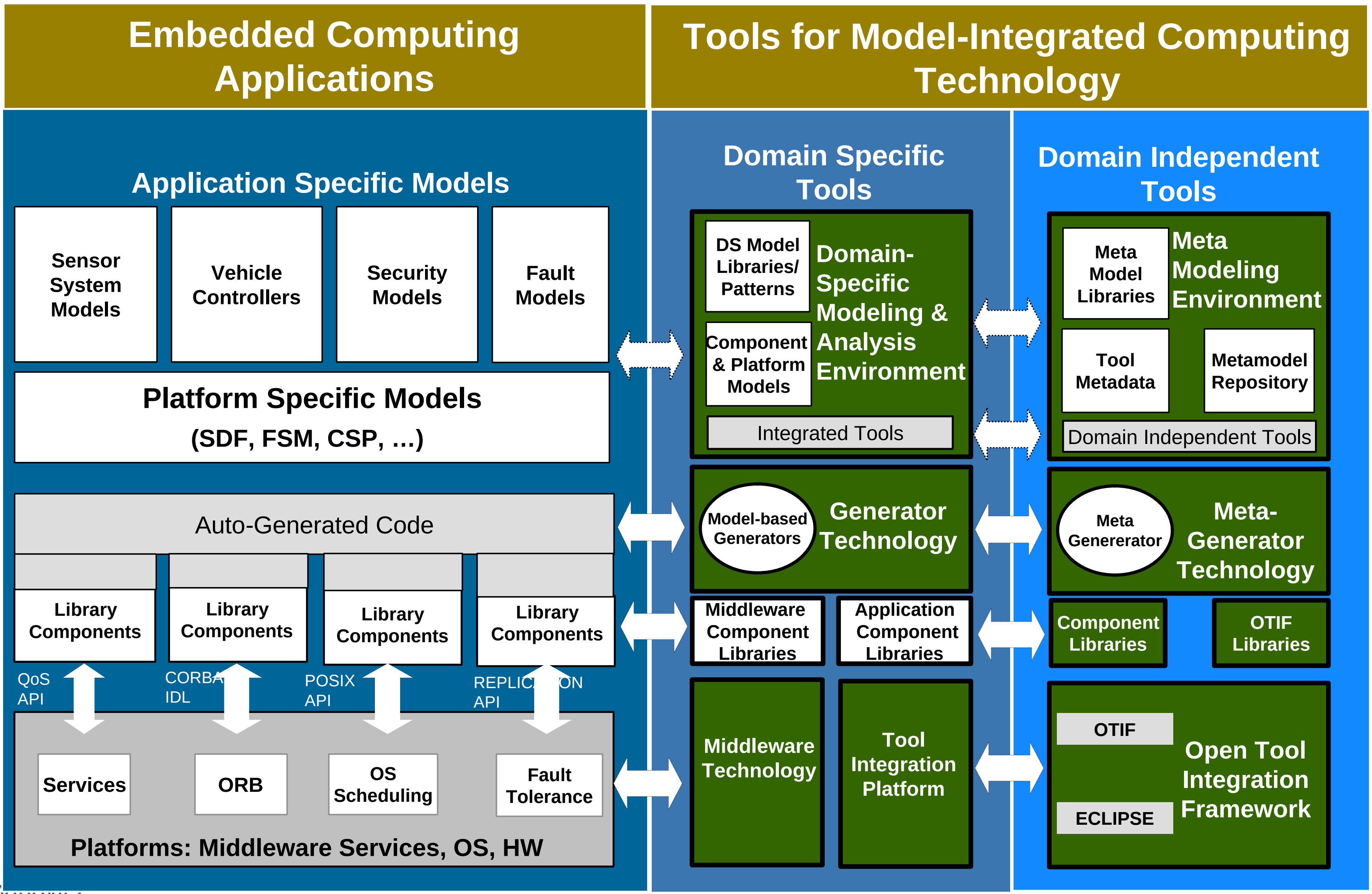
Janos Sztipanovits

July 8,2003

Outline

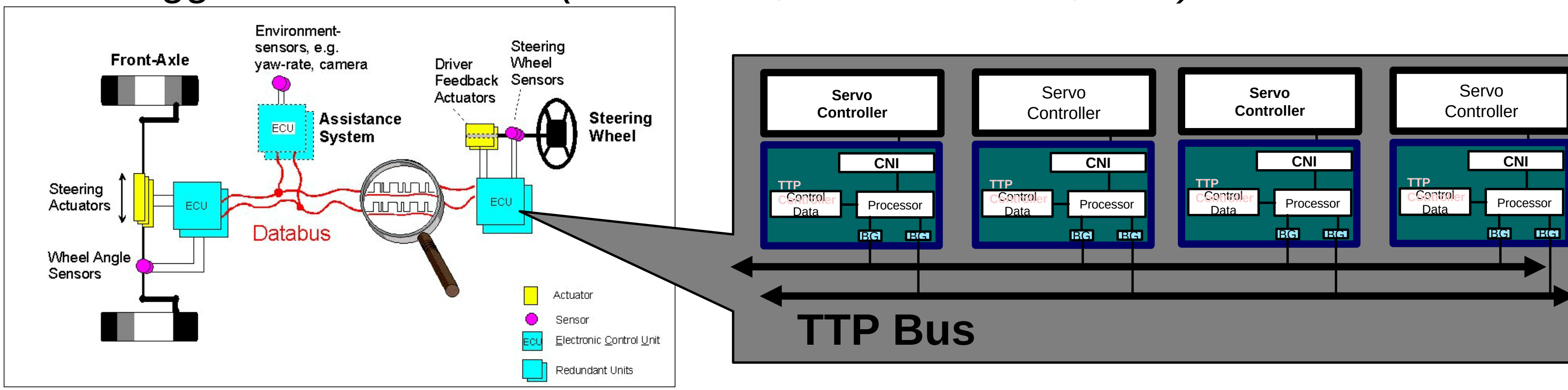
- Model-Integrated Computing
  - Specification of Domain Specific Modeling Languages
  - Specification and Generation of Model Translators
- Model Synthesis Example
  - Construction of Design Spaces
  - Exploration of Design Spaces

The "Grand View" of Model-Integrated Computing



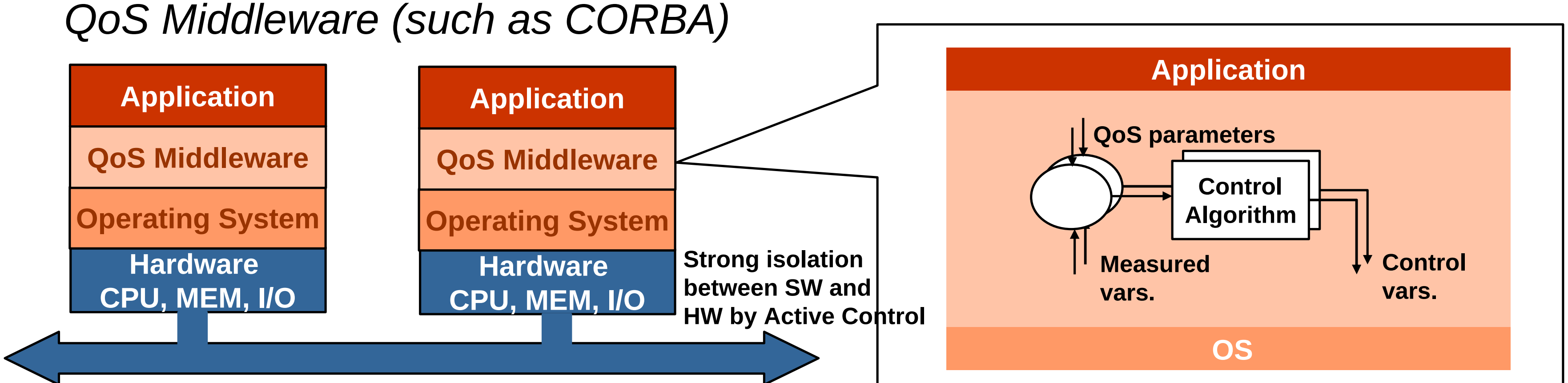
Platforms (There are many…)

Time-Triggered Architecture (distributed, hard real-time, safe)



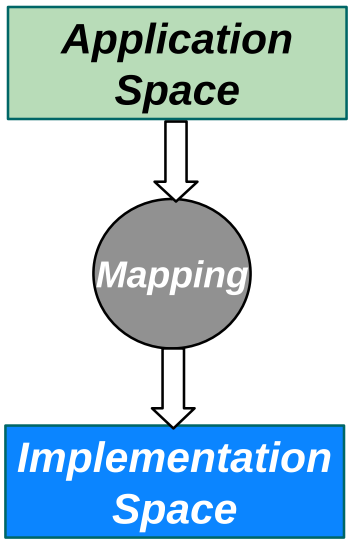
Integration framework, composition mechanisms, components

QoS Middleware (such as CORBA)

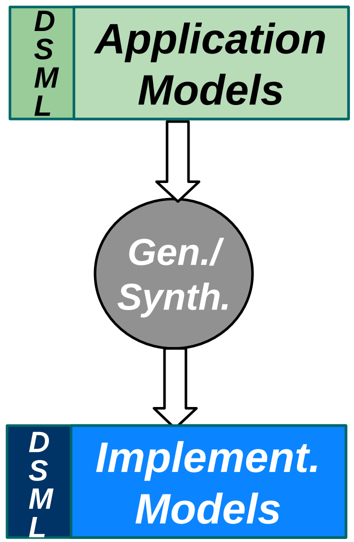


Challenges in Model-Based Design

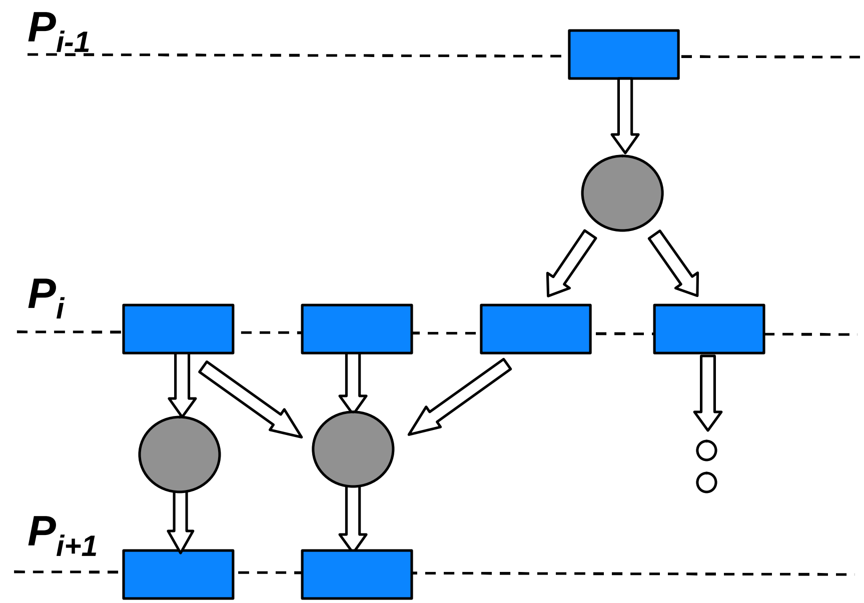
Design



Model-Based Design



Model-Based Design of Embedded Systems

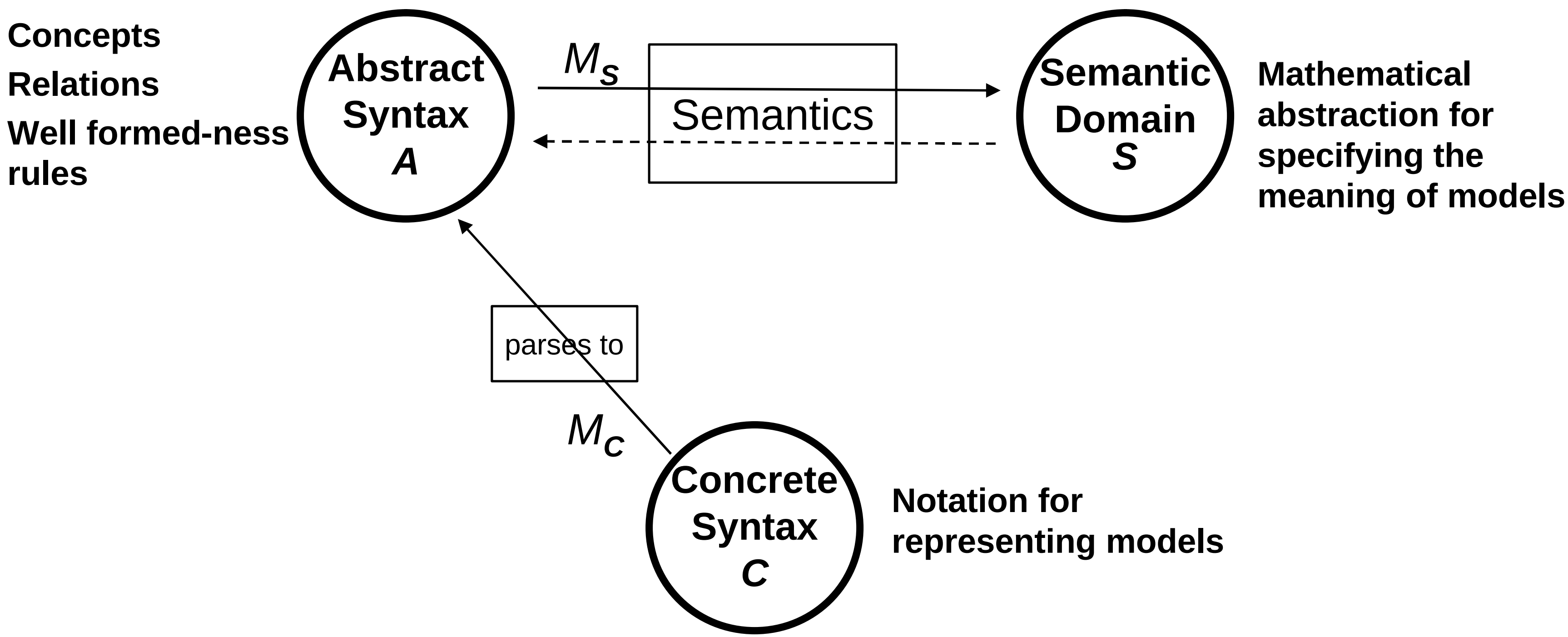


Composition of

- ***Domain Specific Modeling Languages (DSML)***
- ***Model Synthesis***
- ***Model Transformation***

Specification of Domain Specific Modeling Languages (DSML)

$$L = \langle C, A, S, M_S, M_C \rangle$$

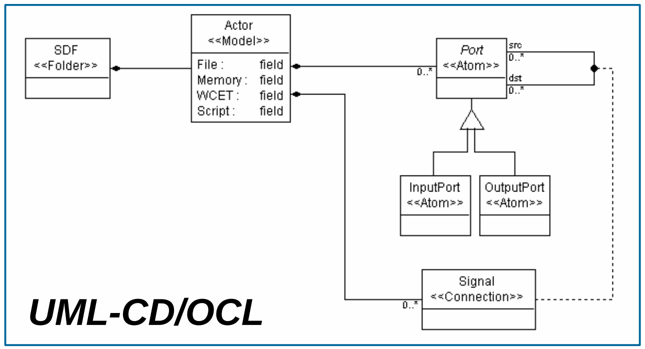


Concrete Syntax and Abstract Syntax

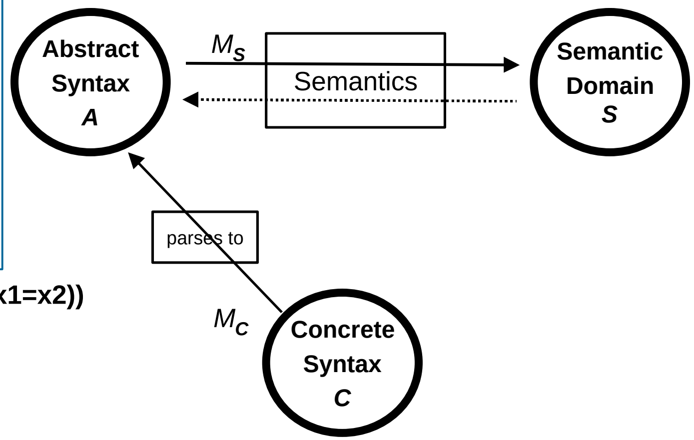
Concepts, Relations

Well formed-ness rules:

Signal Flow Language (SF)

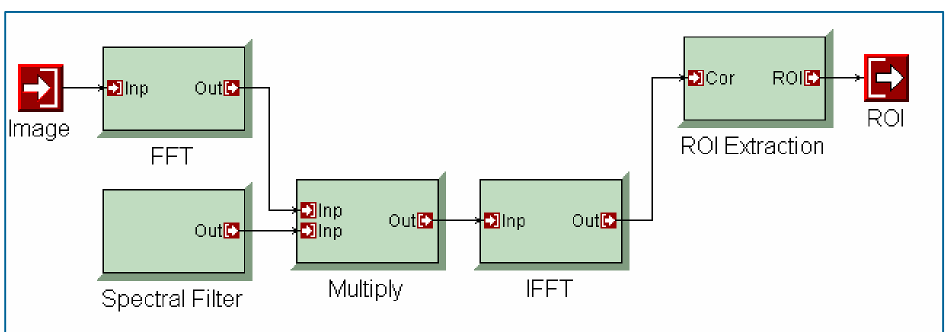


Self.InputPorts()®forAll(ip½ip.src()®forAll(x1,x2½x1=x2))



Mathematical abstraction for specifying the meaning of models

But What About S?

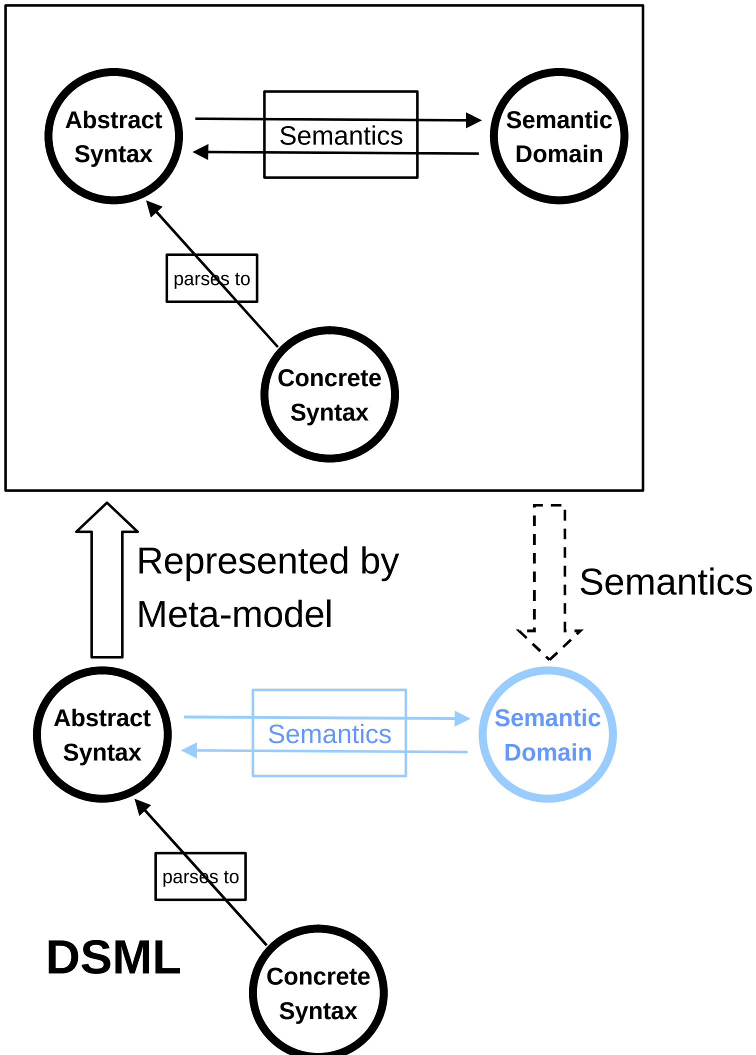


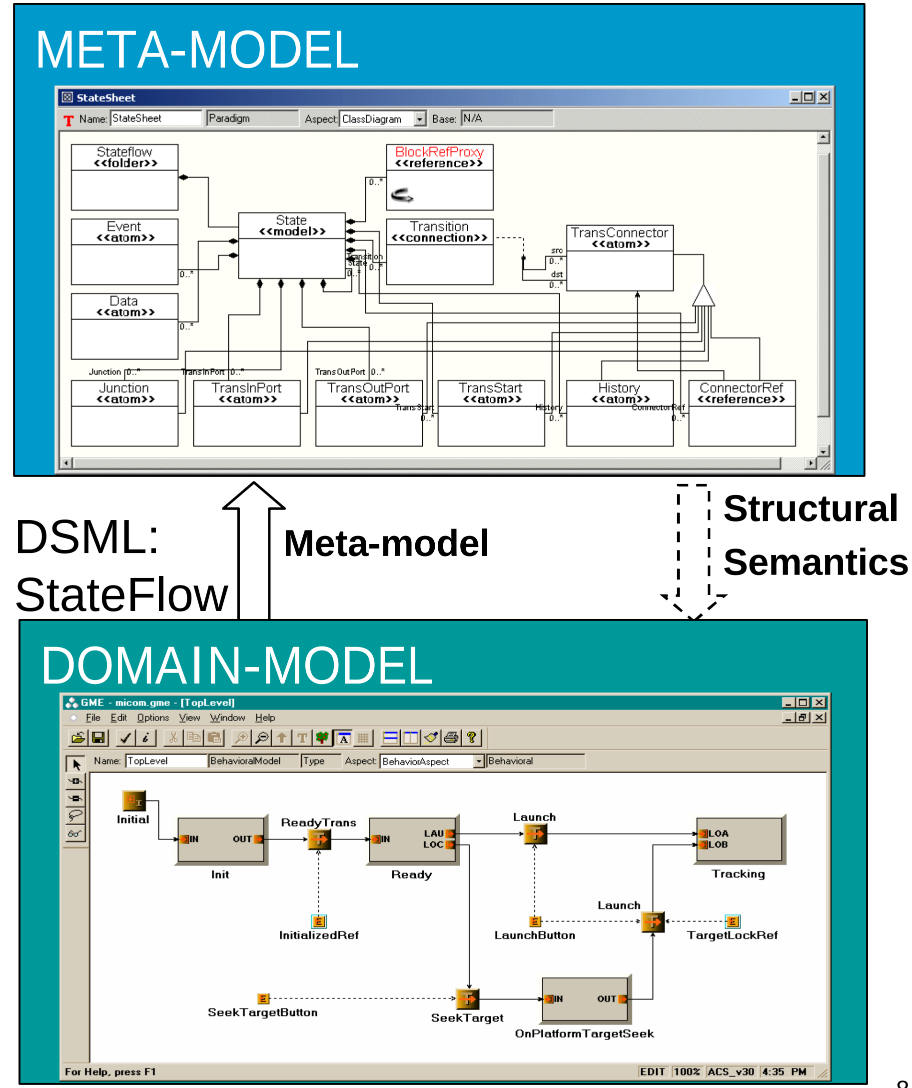
Notation for representing models: E.g.: *Block Diagram*

Semantics via Meta-Modeling

Meta-modeling language with well-defined semantics

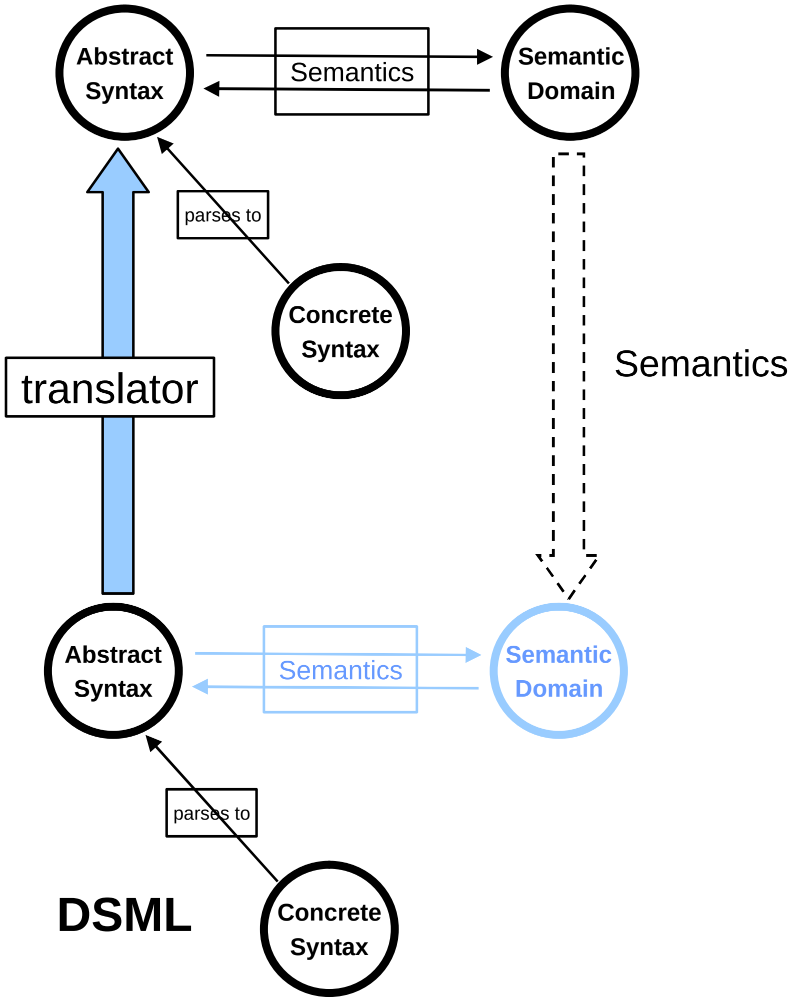
Meta-Model of StateFlow using uml/OCL as meta modeling language.



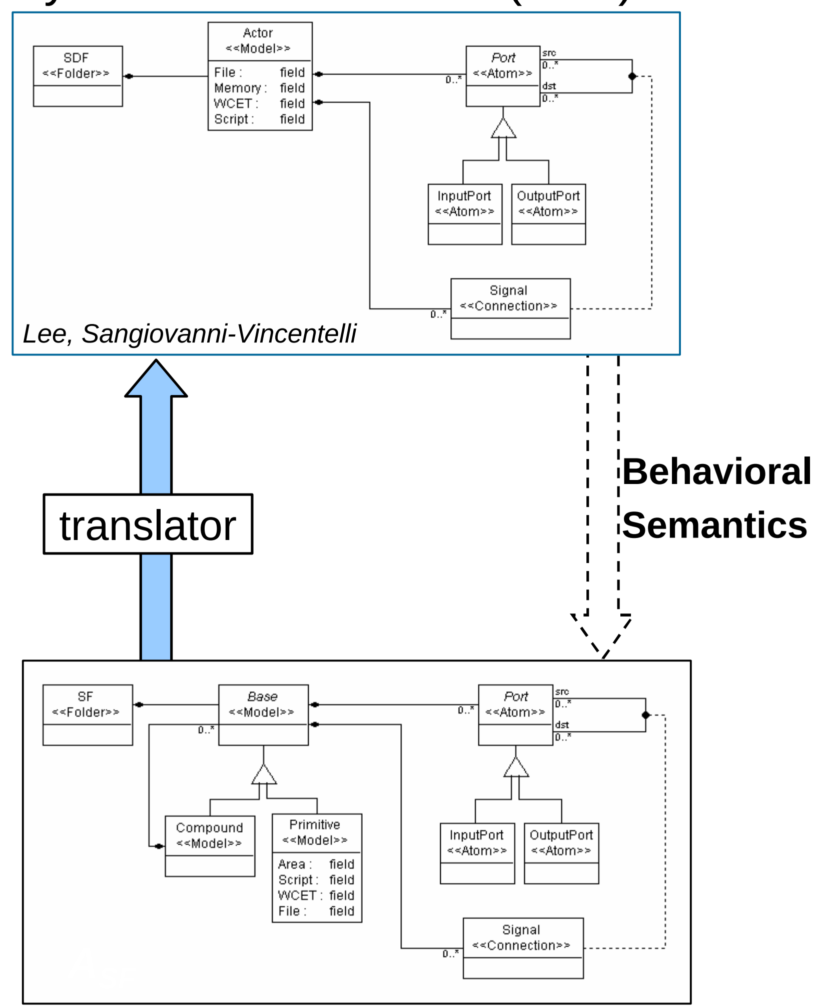


Semantics via Translation

Modeling language with well-defined semantics

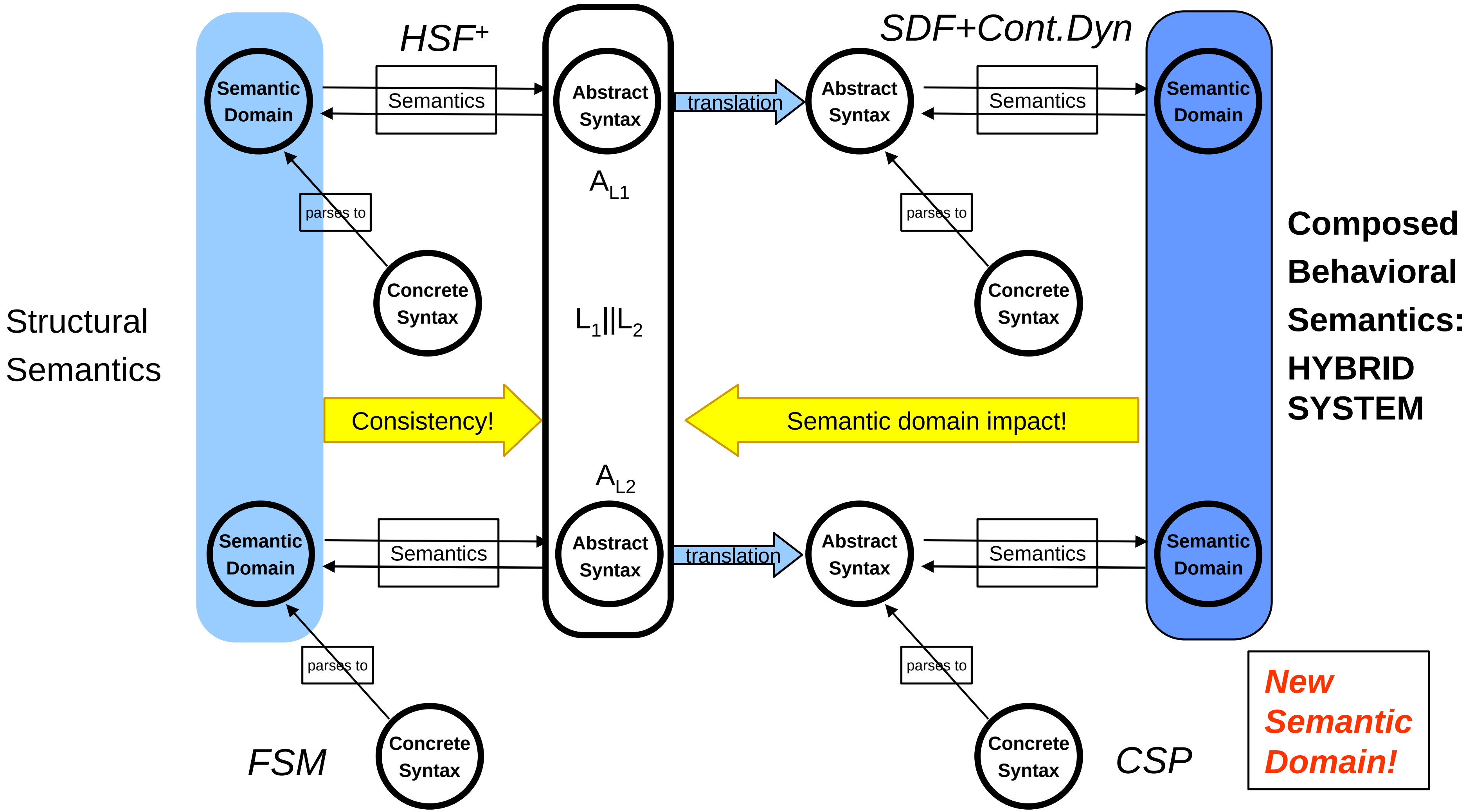


Synchronous Dataflow (SDF)



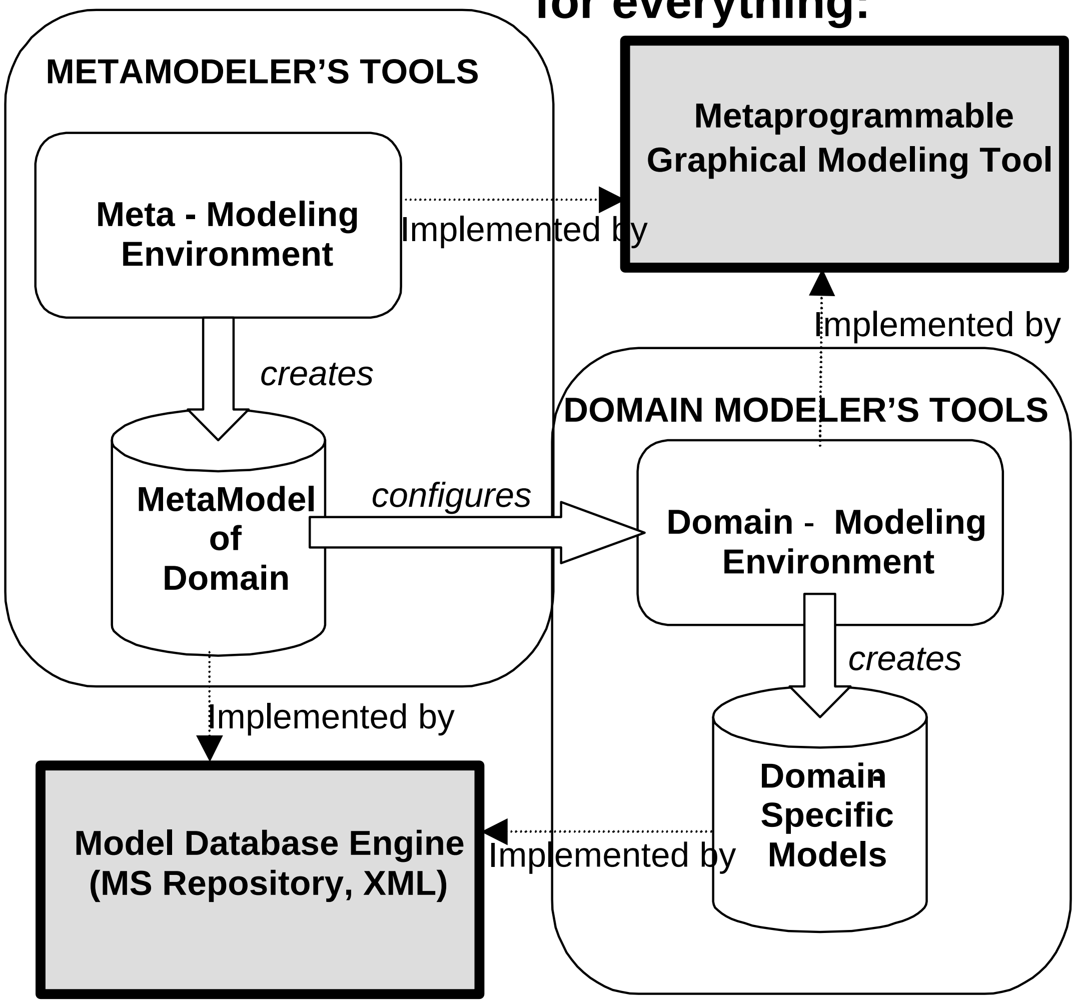
Hierarchical Signal Flow (HSF)

DSML Composition



"Metaprogrammable" Modeling Tool: Generic Modeling Environment (GME)

Using the same core modeling tools for everything:

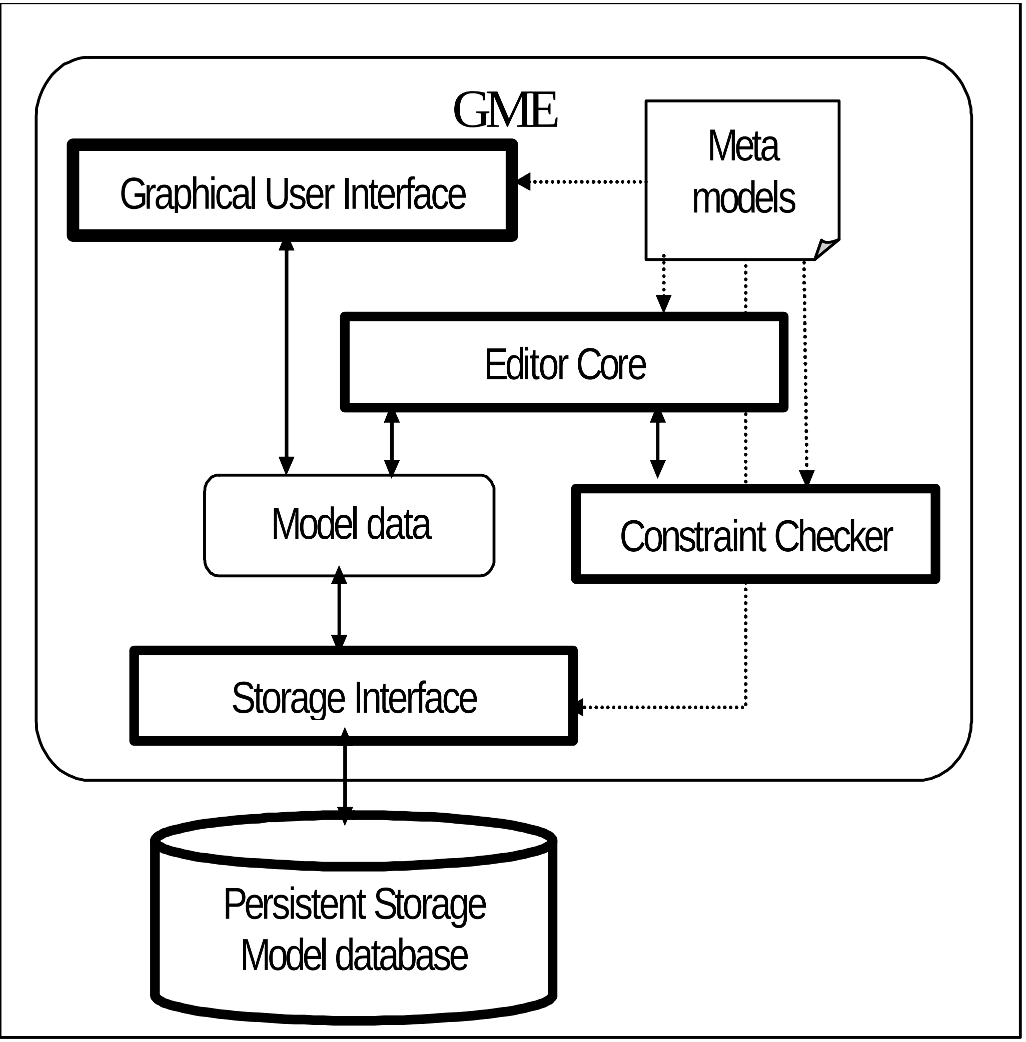


Efficient AND Affordable Modeling

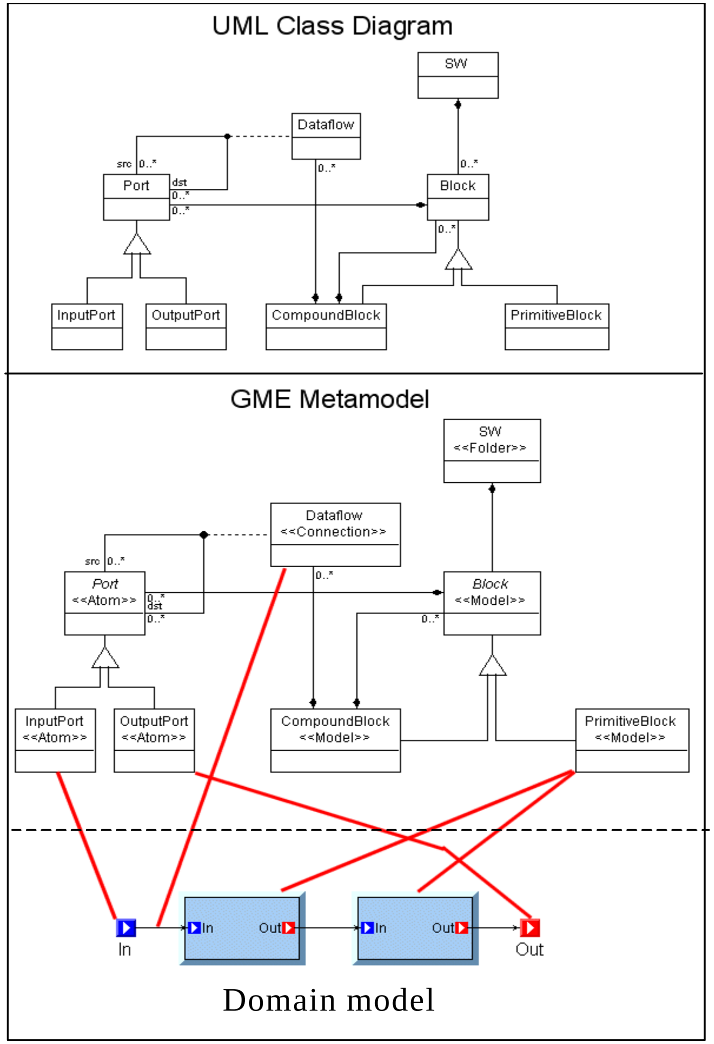
- Model databases, graphical modeling tools are extremely expensive to develop (20-30+ man-year)
- We use COTS and metaprogrammable solutions: domain-specific customization takes only hours
- Opens up new opportunities: "Design your domain specific modeling language for your R&D program and configure the modeling and model repository tools using libraries."

GME Architecture

Internal Structure



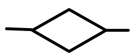
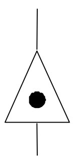
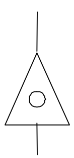
Use



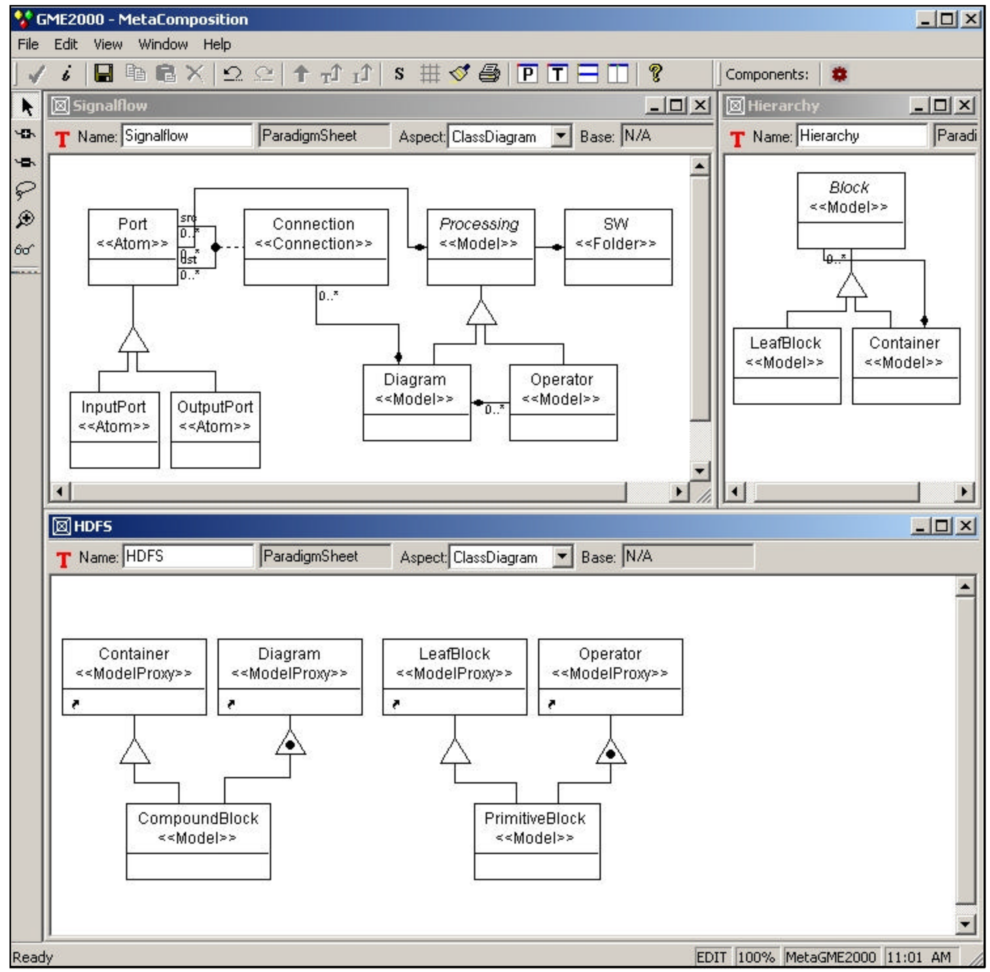
[download site: http://www.isis.vanderbilt.edu/Projects/gme]

GME Support for Compositional Meta-Modeling

Composition Operators

Operator	Symbol	Informal semantics
Equivalence		Complete equivalence of two classes
Implementation Inheritance		Child inherits all of the parent's attributes and those containment associations where parent functions as container.
Interface Inheritance		Child inherits all associations except containment associations where parent functions as container.

Metamodel composition with GME



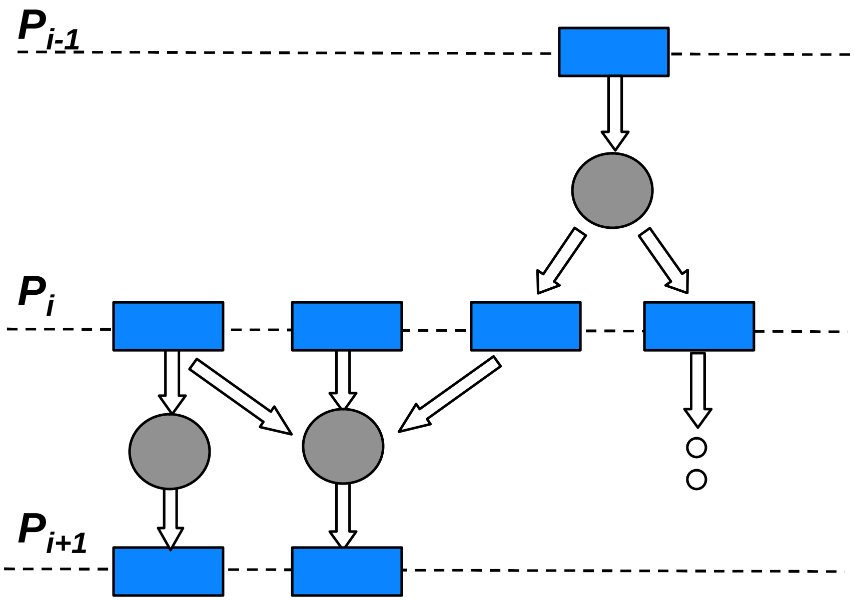
Research Issues in Domain Specific Modeling Languages

- **Precise, compositional meta-modeling**
- **Multiple aspect modeling in the compositional meta-modeling framework**
- **Practical issues:**
  - Examples, meta-model libraries
  - Meta-programmable tools
  - Link to UML-2

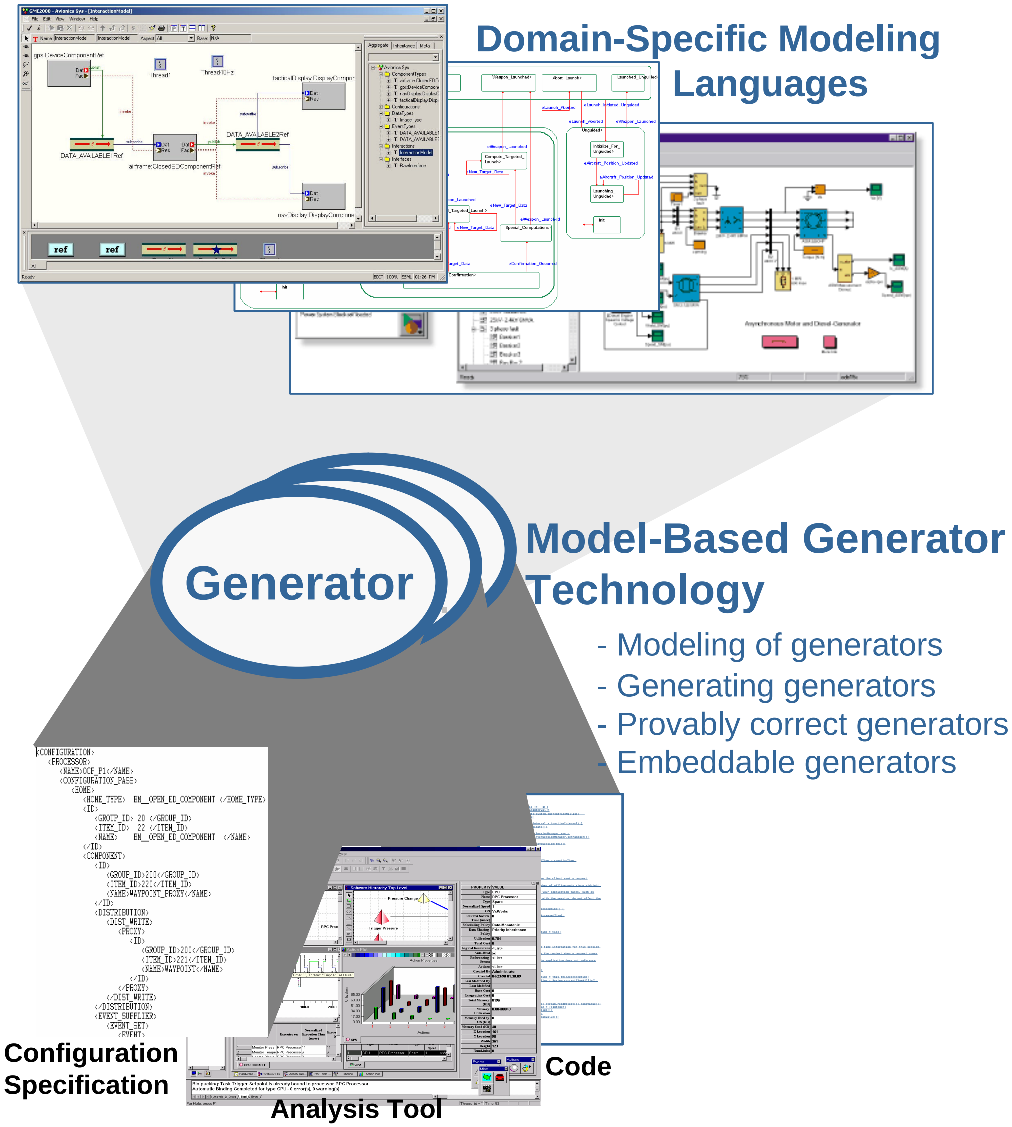
Model Synthesis and Transformations

VANDERBILT UNIVERSITY
INSTITUTE FOR SOFTWARE INTEGRATED SYSTEMS

Model-Based Design of Embedded Systems



- ***Model Synthesis***
- ***Model Transformation***



Generator Technology in MIC

1. **Direct implementation**
   - Input tree generation (implicit: models ARE the input tree)
   - Traverse input tree and incremental construction of output tree
   - Printing out the “product”
   - Widely used in MIC, good tool support
2. **Pattern-based design**
   - “Visitor pattern”
   - Explicit specification of traversal paths
   - See Karsai and Lieberherr

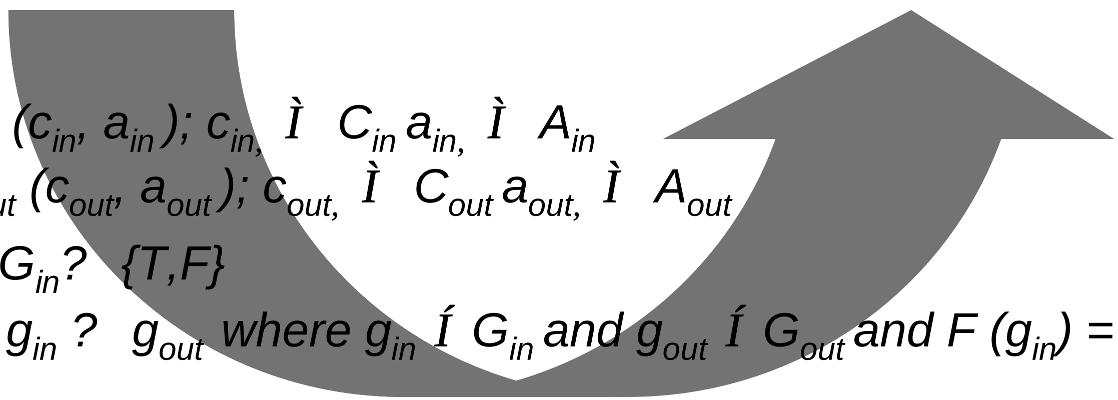
Generator Technology in MIC

3. Meta Generators (Karsai)

- Formal modeling of generators and generating the generators from the formal models
- Formalism: graph grammars and graph rewriting

Input graph: G_{in} (C_{in}, A_{in})

Output graph: G_{out} (C_{out}, A_{out})



g_{in} (c_{in}, a_{in}); $c_{in,}$ Ì C_{in} $a_{in,}$ Ì A_{in}

g_{out} (c_{out}, a_{out}); $c_{out,}$ Ì C_{out} $a_{out,}$ Ì A_{out}

F: G_{in}? {T,F}

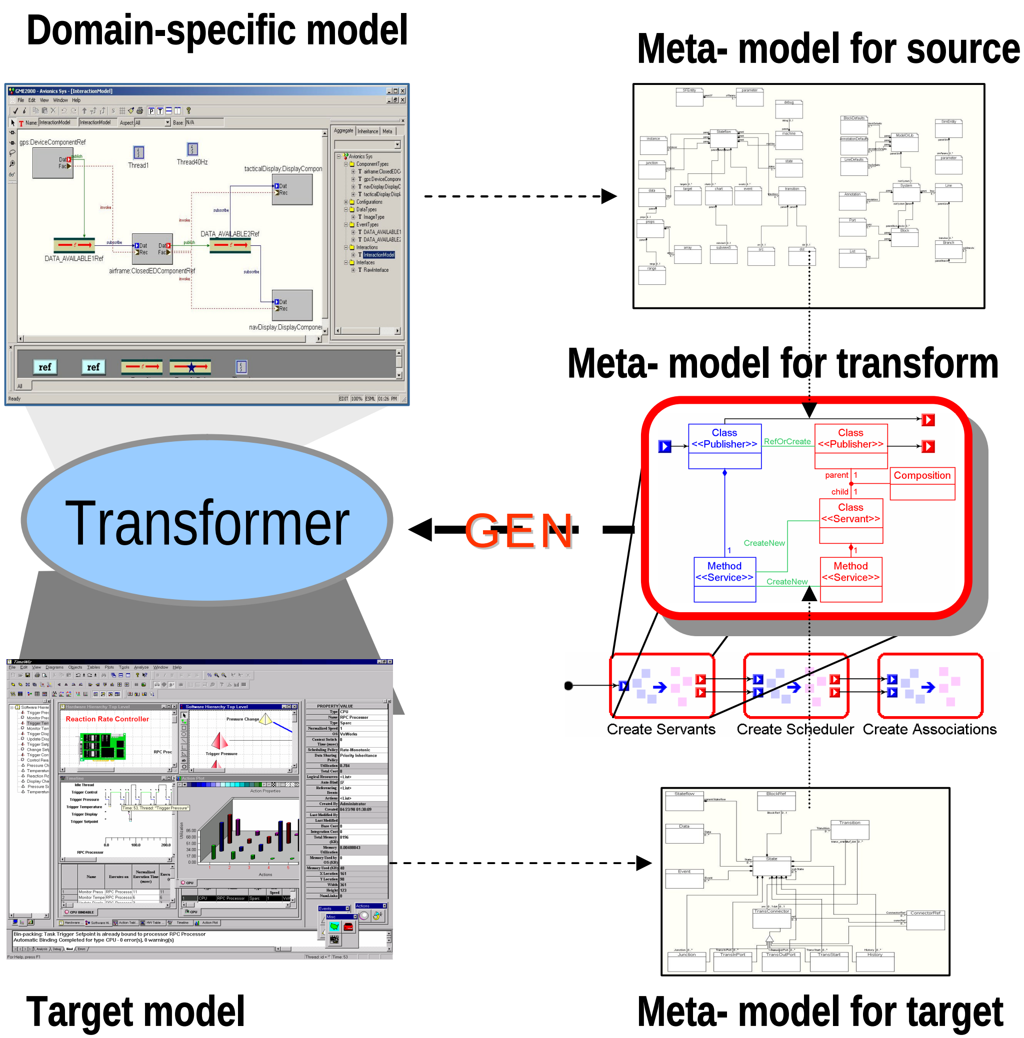
M: g_{in} ? g_{out} where g_{in} Í G_{in} and g_{out} Í G_{out} and F (g_{in}) = T

Meta-generators: *Model Transformations in Tool Integration*

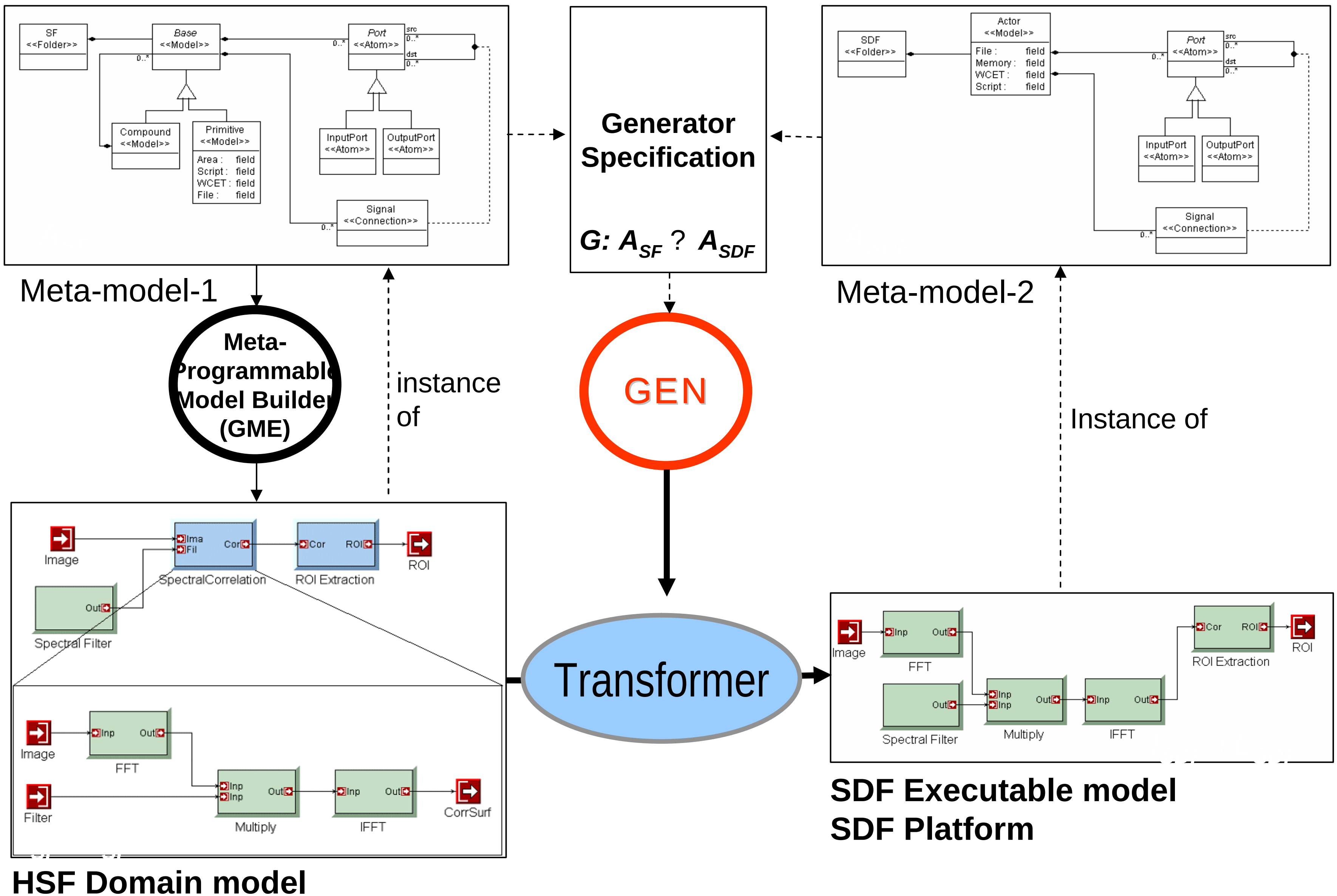
<u>Roles transformations play in model-based design:</u>

- Refining a design into an implementation
- Code generation
- PIM -> PSM mapping
- Support for model interchange for tool integration

Approach: Meta-models for source and target models plus transformations, then <u>generating the transformer</u>



Meta-generators: *Model Transformations in Component Integration*

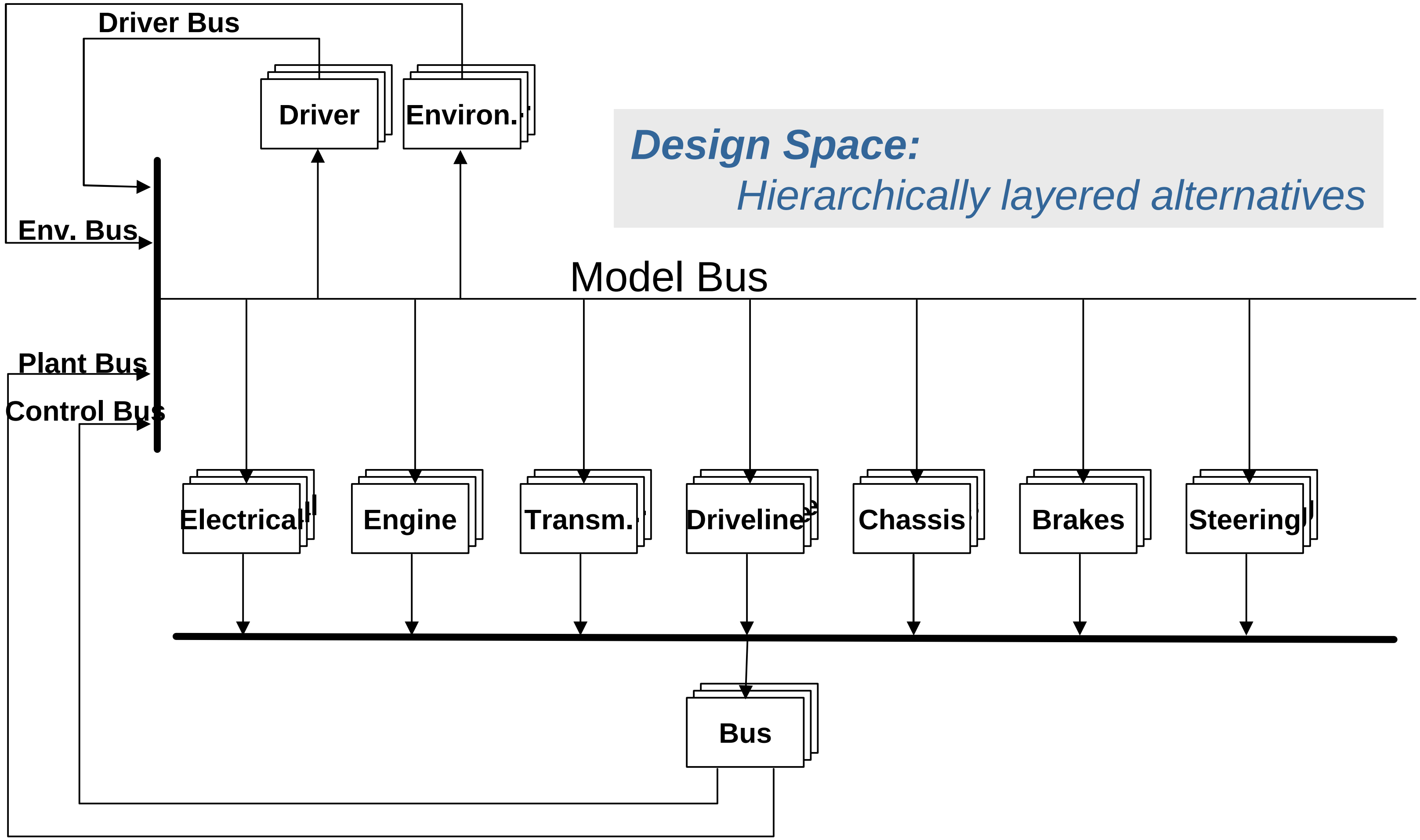


Research Issues in Model Transformations

- Languages and tools for meta generators
- Model synthesis using explicit design patterns
- Model synthesis using constraint-based design-space exploration
- Generative modeling extensions to languages
- Embeddable generators

Example: Constraint-based Model Synthesis

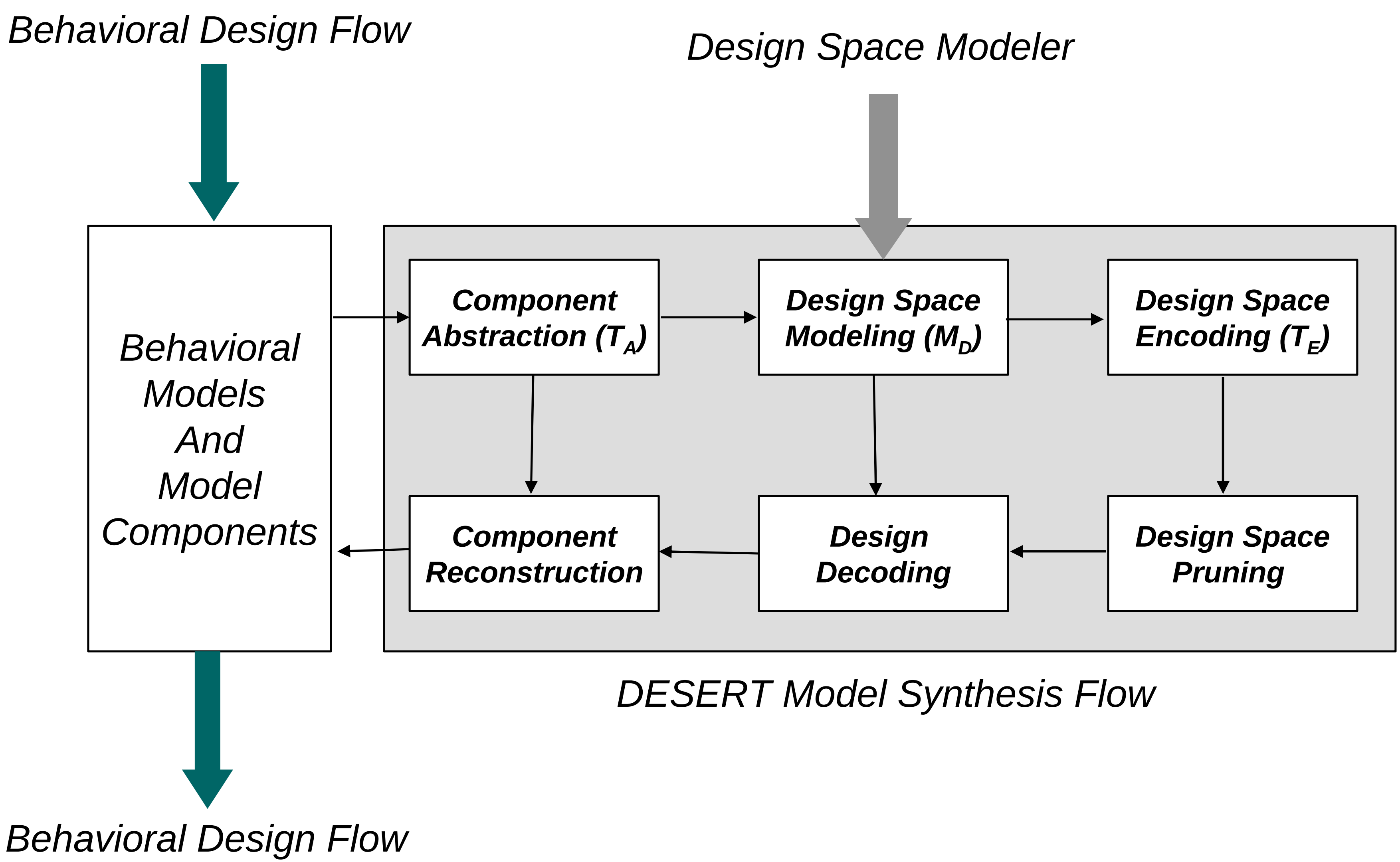
A modeling problem:



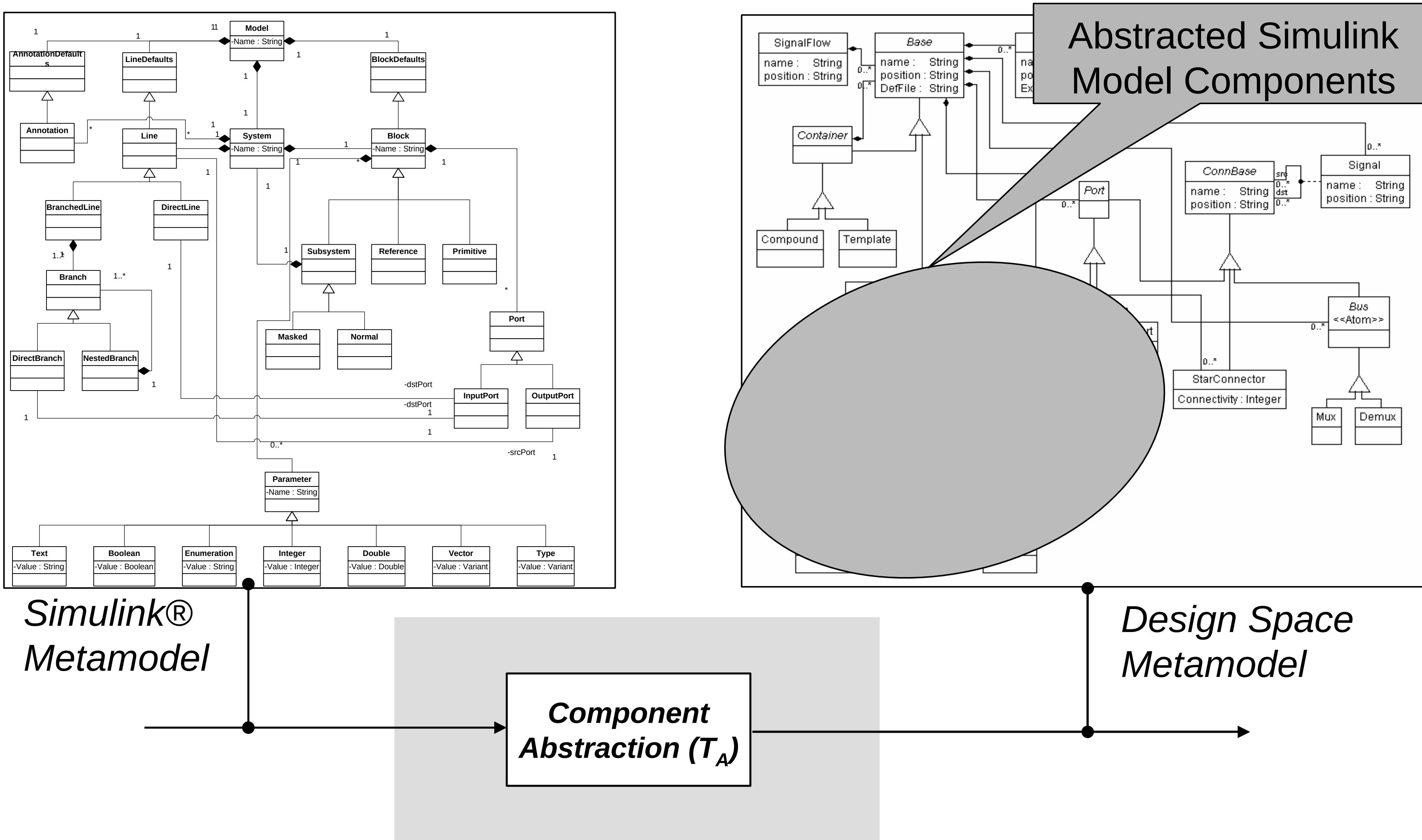
Design Space:
Hierarchically layered alternatives

Source: Ken Butts, Ford

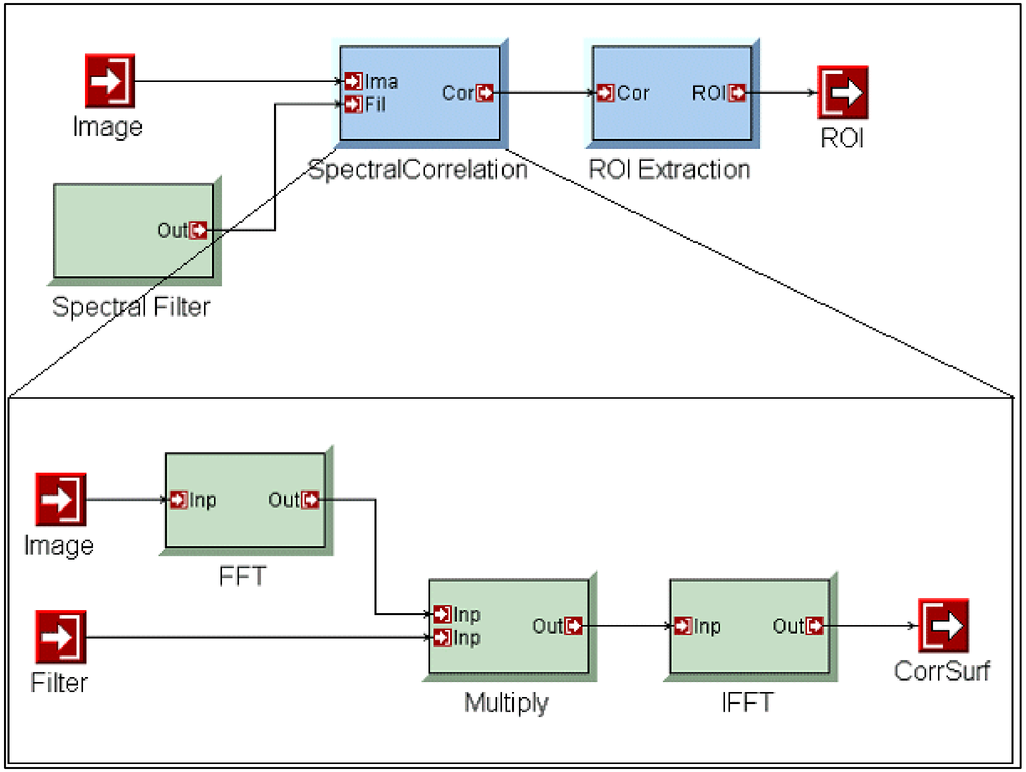
Reusable Model Synthesis

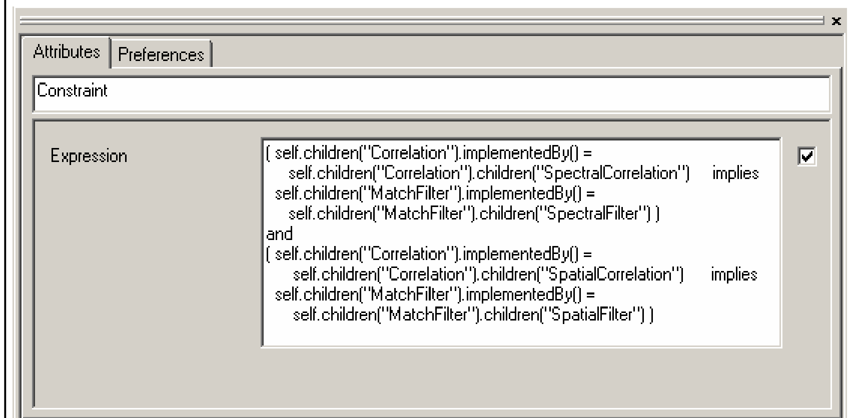


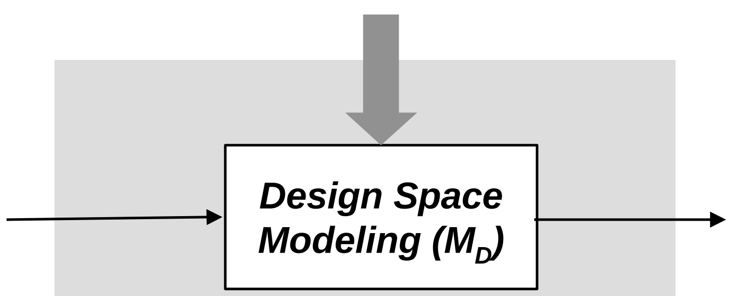
Component Abstraction



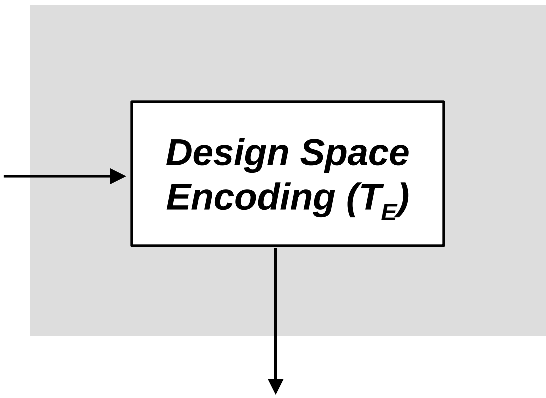
Design Space Modeling

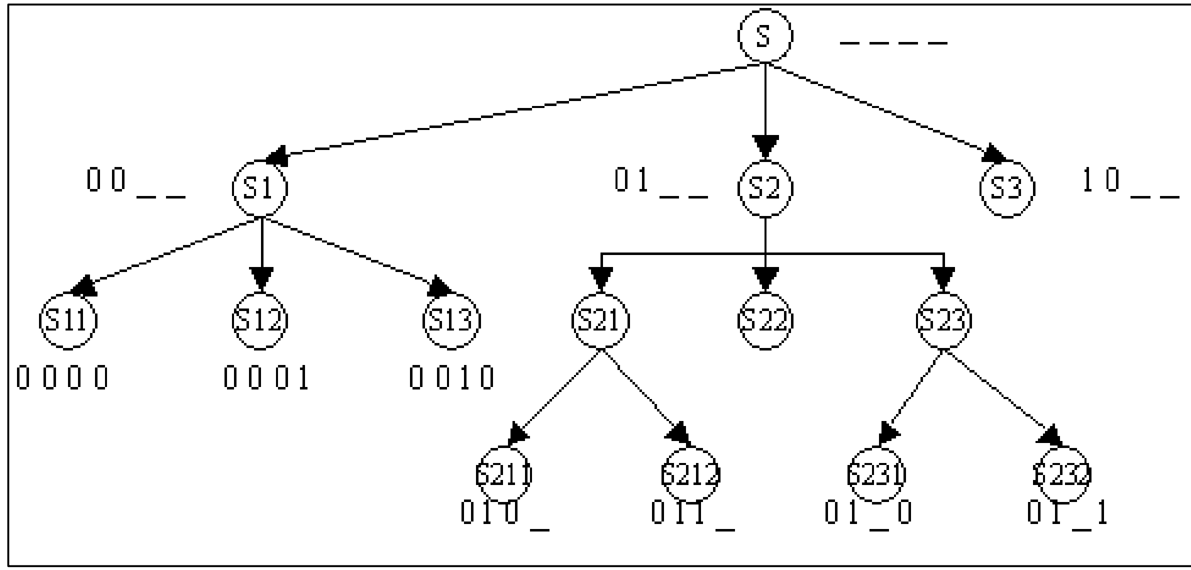






Design Space Encoding



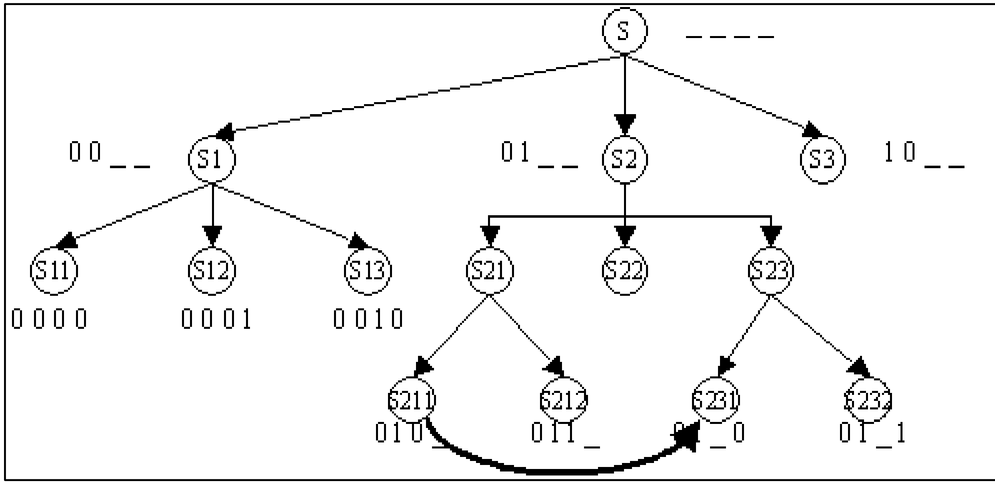


$$S = S_1 \vee S_2 \vee S_3$$
$$S_1 = S_{11} \vee S_{12} \vee S_{13}$$
$$S_2 = S_{21} \wedge S_{22} \wedge S_{23}$$
$$S_{21} = S_{211} \vee S_{212}$$
$$S_{23} = S_{231} \vee S_{232}$$

$$S_{11} = \neg v_0 \neg v_1 \neg v_2 \neg v_3$$
$$S_{12} = \neg v_0 \neg v_1 \neg v_2 v_3$$
$$S_{13} = \neg v_0 \neg v_1 v_2 \neg v_3$$
$$S_{22} = \neg v_0 v_1$$
$$S_3 = v_0 \neg v_1$$

$$S_{211} = \neg v_0 v_1 \neg v_2$$
$$S_{212} = \neg v_0 v_1 v_2$$
$$S_{231} = \neg v_0 v_1 \neg v_3$$
$$S_{232} = \neg v_0 v_1 v_3$$

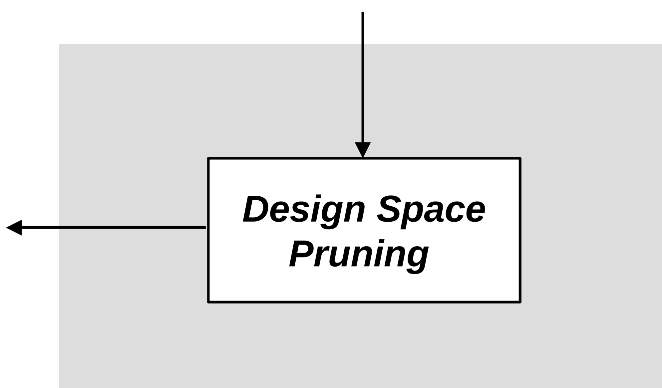
Design Space Pruning



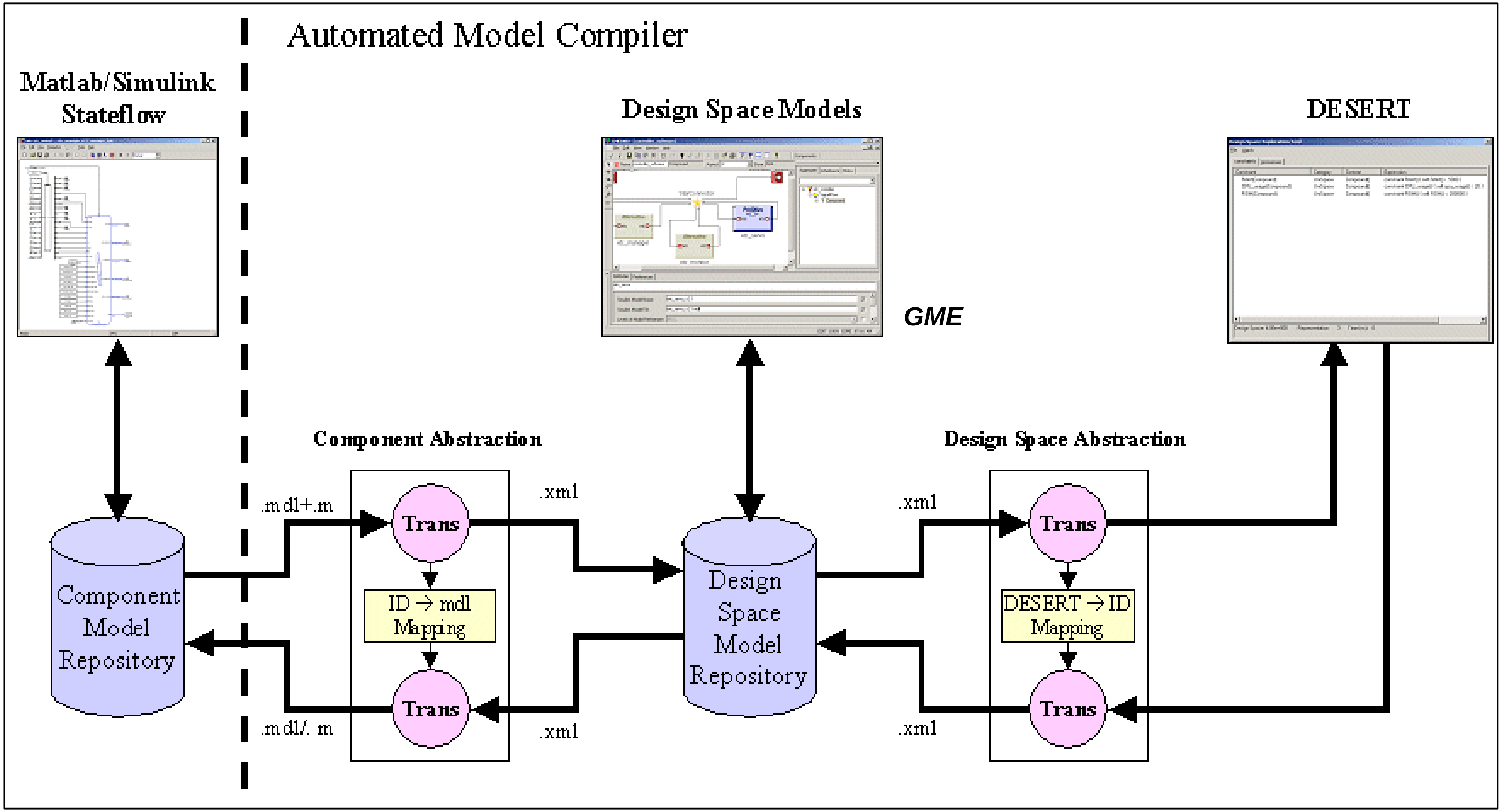
$$cc = S_{211} \Rightarrow S_{231}$$

$$cc = S_{211} \wedge S_{231} \vee \neg S_{211}$$

$$cc = \neg v_0 v_1 \neg v_2 \neg v_3 \vee v_0 \vee \neg v_1 \vee v_2$$



DESERT Tool Chain Summary



Conclusion

- The hard problems of building large embedded systems are Semantics and Compositionality
- Model-based integration technology has the power to solve the problem
- Composable DSMLs and model transformations are key for tool reuse.