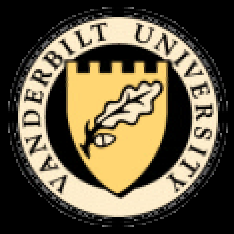


Institute for Software Integrated Systems

Vanderbilt University

Specifying Graphical Modeling Systems Using Constraint-based Metamodels

Gabor Karsai, Greg Nordstrom,
Akos Ledeczi, Janos Sztipanovits

Presented by Greg Nordstrom

IEEE International Symposium on Computer-Aided Control System Design

Anchorage, Alaska

September 25-27, 2000

Overview

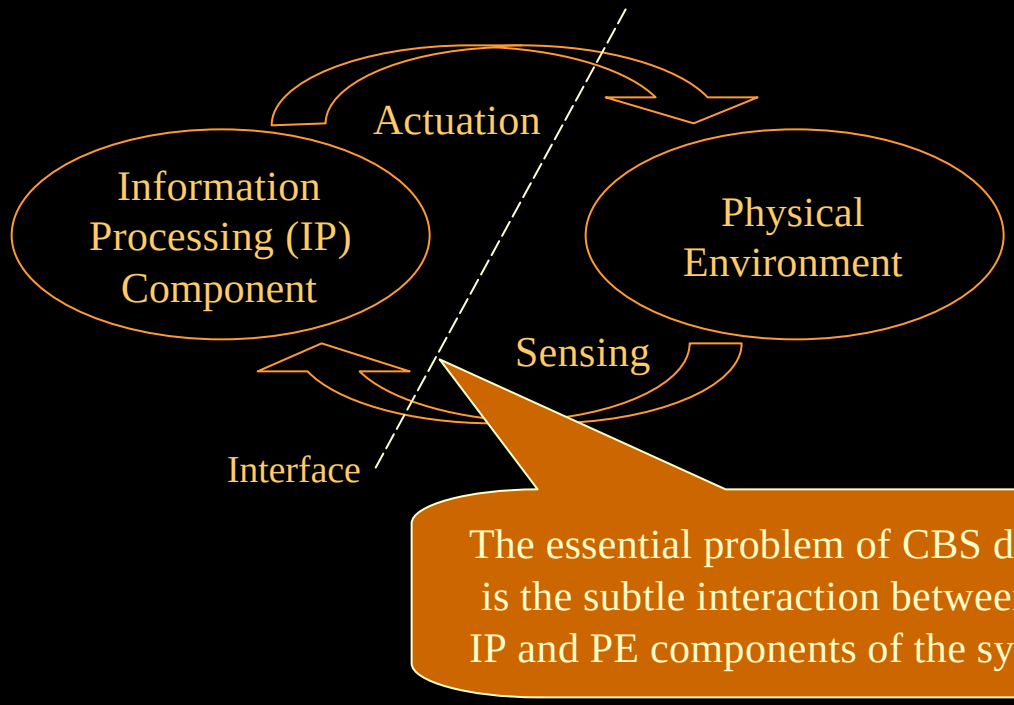
- Computer-based system (CBS) design
  - The argument for domain-specific modeling
- Modeling language definition
  - Language components
- The meta-level modeling approach
  - Metamodel specification
  - Advantages of the approach
- Conclusions and future work

Computer-Based Systems

- CBS characteristics
  - Hardware and software tightly coupled
  - Dynamic operating environment
  - Critical to the enterprise
  - Long lifespan
  - Multi-disciplinary development
- Examples
  - Embedded systems
  - Process monitoring, analysis
  - Manufacturing execution systems

CBS design is highly non-trivial

Computer-Based Systems



System behavior determined by:

- HW, SW of IP component
- Interfaces to physical environment
- Physical environment itself
- System-level constraints

An *integrated* approach is needed to develop the engineering science of such systems, where *all* aspects of the design can be analyzed.

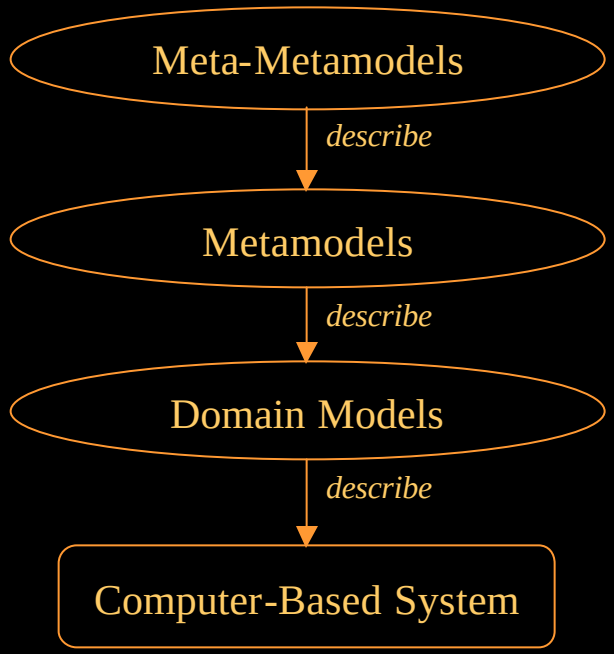
Arguments and Proposals

- No one modeling language satisfies the requirements of all CBS's
- No single engineering discipline exists for CBS design
- Common language should be that of the *domain*, not computer engineering
- Modeling tools must be rich enough to support capture of IP-PE interaction
- Integration of models and tools is necessary for CBS analysis and synthesis
- Modeling tool evolution must be easy, predictable, and safe

We propose a **two-level approach**:

1. Domain-specific modeling tools for creating domain-specific models
2. Modeling tools represented by, *and built from*, metamodels

The Four Layers of Modeling



The meta-language is not used for defining domain *models*, but rather for defining domain-modeling *languages*.

Defining a Modeling Language

A modeling language $L=<O, S, I>$ consists of:

Ontology, *O*	Concepts and their relationships in the language
Syntax, *S*	Defines syntactically correct sentences in the language
Interpretation, *I*	Semantics—the meaning of those correct sentences. This includes both *static* and *dynamic* semantics.

e.g.:

Meta-level language, L_M

$L_M=<O_M, S_M, I_M>$

Domain-specific modeling language, L_D

$L_D=<O_D, S_D, I_D>$

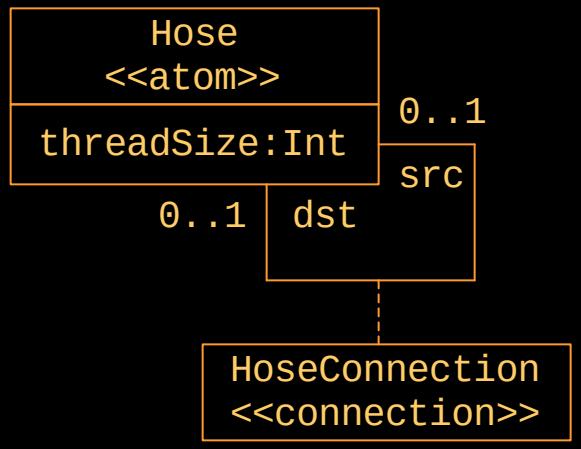
Metamodels define L_D in terms of $<O_M, S_M, I_M>$. This implies that a meta-language must allow us to define *ontologies*, *syntax*, and *interpretation* in a mathematically precise way.

Metamodel Ontology

- L_M must allow for definition of modeling concepts used to define systems within a particular domain
  - Instances of concepts and relationships defined in O_M define L_D
- Set of fundamental modeling abstractions exists
  - Attributed classes and entities (entities may contain other entities)
  - Binary, n-ary associations between classes and entities
  - Hierarchy (“aggregation through containment”)
  - Specialization/generalization (“inheritance”)
  - Constraints (binary expressions of invariance)
  - Module interconnection
  - Multiple aspects

Specifying Syntax and Static Semantics

- UML class diagrams + OCL expressions (constraints)

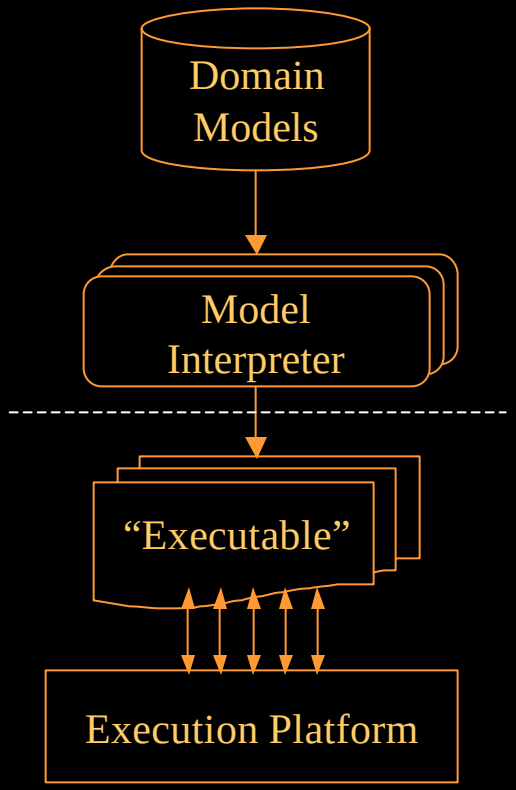



```
// connected hoses must have
// same sized threads
HoseConnection.allInstances ->
  forAll(c| c.src.threadSize =
            c.dst.threadSize)

// can't connect hose to itself
HoseConnection.allInstances ->
  forAll(c| c.src <> c.dst)
```

- Additional non-UML syntactical constructs define:
  - Modeling abstractions not easily expressed in UML (e.g. *Aspects*)
  - Visualization information (i.e. presentation specifications)

Assigning Dynamic Semantics to Domain Models



Two-phase approach:

1. *Transformation* phase
   - Model interpreters (e.g. *VC++*, *VB*, *C#*) transform models into "instruction set" of execution platform

2. *Execution* phase
   - Execution platform executes "instructions"

Advantages

- Computer-aided metamodel validation
- Synthesis of domain-specific modeling tools
  - Domain-specific modeling tools yield only valid domain models
- Multi-domain model integration
  - Controlled integration of existing tools
  - Metamodel specifies model integration syntax and semantics
- Rapid, predictable modeling environment evolution
  - Modeling tools remain current
  - Effective framework for model migration

Conclusions and Future Work

- DSMLs can be…
  - …described with mathematical precision
  - …safely evolved over time
- Metamodels can be…
  - …machine validated
  - …used to synthesize integrated, domain-specific modeling tools
- Domain-specific models can be…
  - …“executed” on various execution platforms
  - …migrated to new environments over time
- Next…
  - Libraries of metamodel solutions to “standard” modeling problems
  - Formal interpreter specifications in metamodels
  - XML-based metamodel storage, retrieval