Capturing Domain-Specific Knowledge for Design of Hydraulic Systems

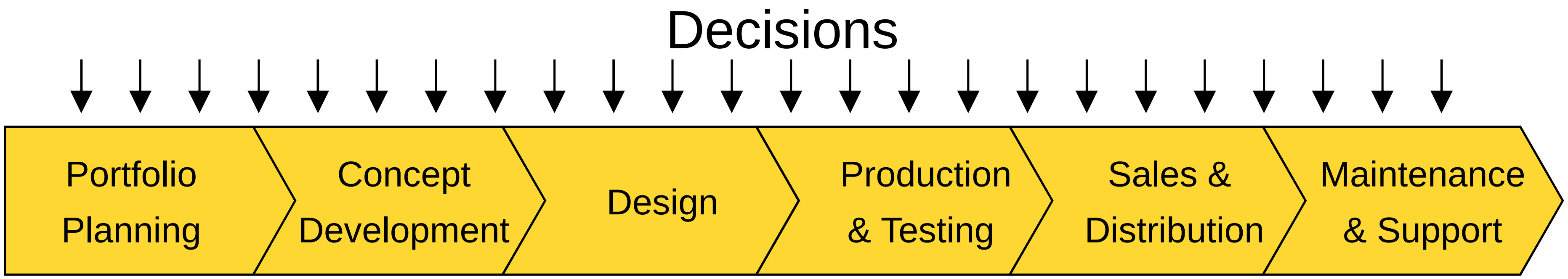
Systems **R**ealization **L**aboratory

Chris Paredis

Systems Realization Laboratory
Product and Systems Lifecycle Management Center
G.W. Woodruff School of Mechanical Engineering
Georgia Institute of Technology

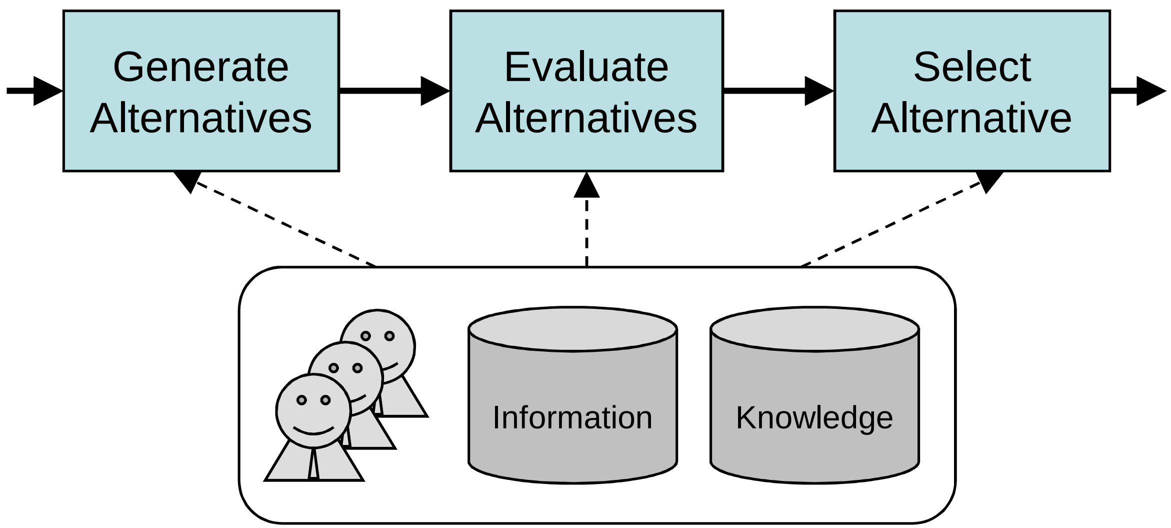
www.srl.gatech.edu www.pslm.gatech.edu

Systems Engineering: A Decision-Based Perspective



Modeling and Simulation Provides Information in Support of Decisions

Generic Decision Process



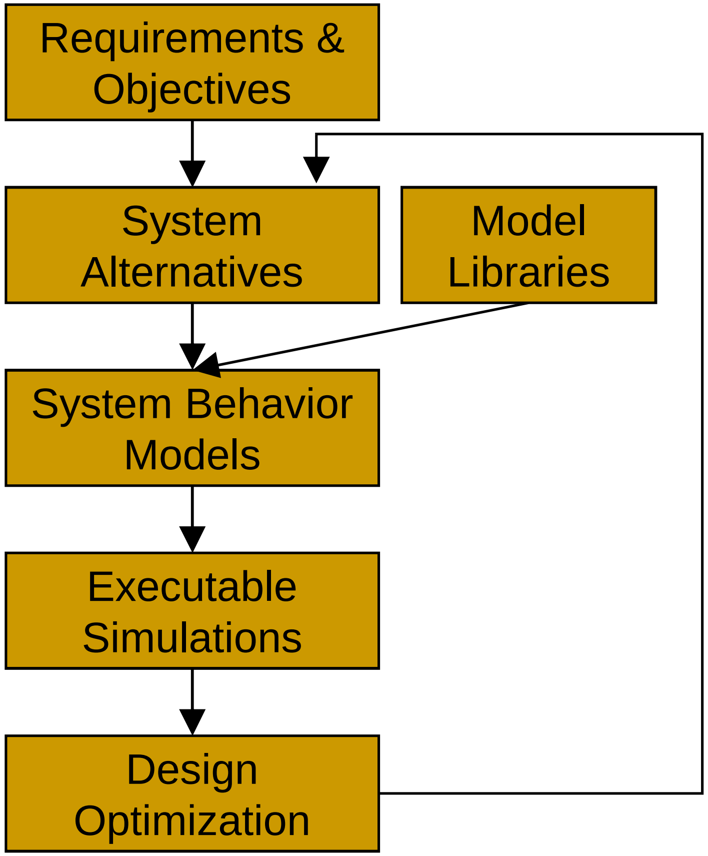
Challenges in Systems Engineering

- Multiple integrated functions
- Multiple engineering disciplines
- Multiple stakeholders
- Globally distributed, heterogeneous design teams
- Complex, emergent system behavior
- Large quantities of design knowledge and information

→ Need Formal, Model-Based Approach

Model-based Systems Engineering (MBSE)

MBSE: Model formally all aspects of a systems engineering problem



- Effective and Efficient Analysis of Alternatives
  - Model from different perspectives
  - Model at different levels of abstraction
  - Variable-fidelity modeling
  - Model reuse & modularity
- Effective Generation of Alternatives
  - Graph transformations for generating plausible system architectures
  - Automated generation of system models

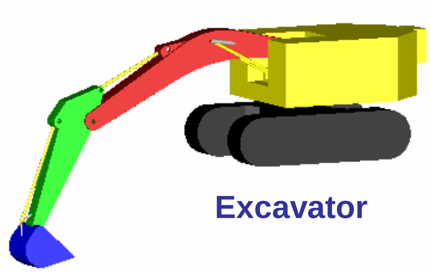
MBSE Example Problem: Hydraulic Systems

Given:

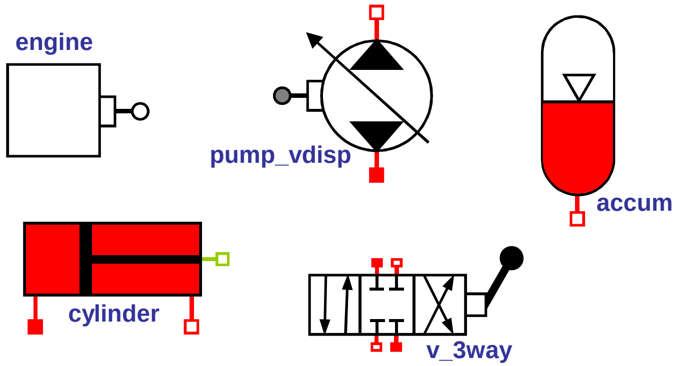
- Primary components
- Decision objectives / preferences

Find:

- Best system topology
- Best component parameters



How to size and connect these?

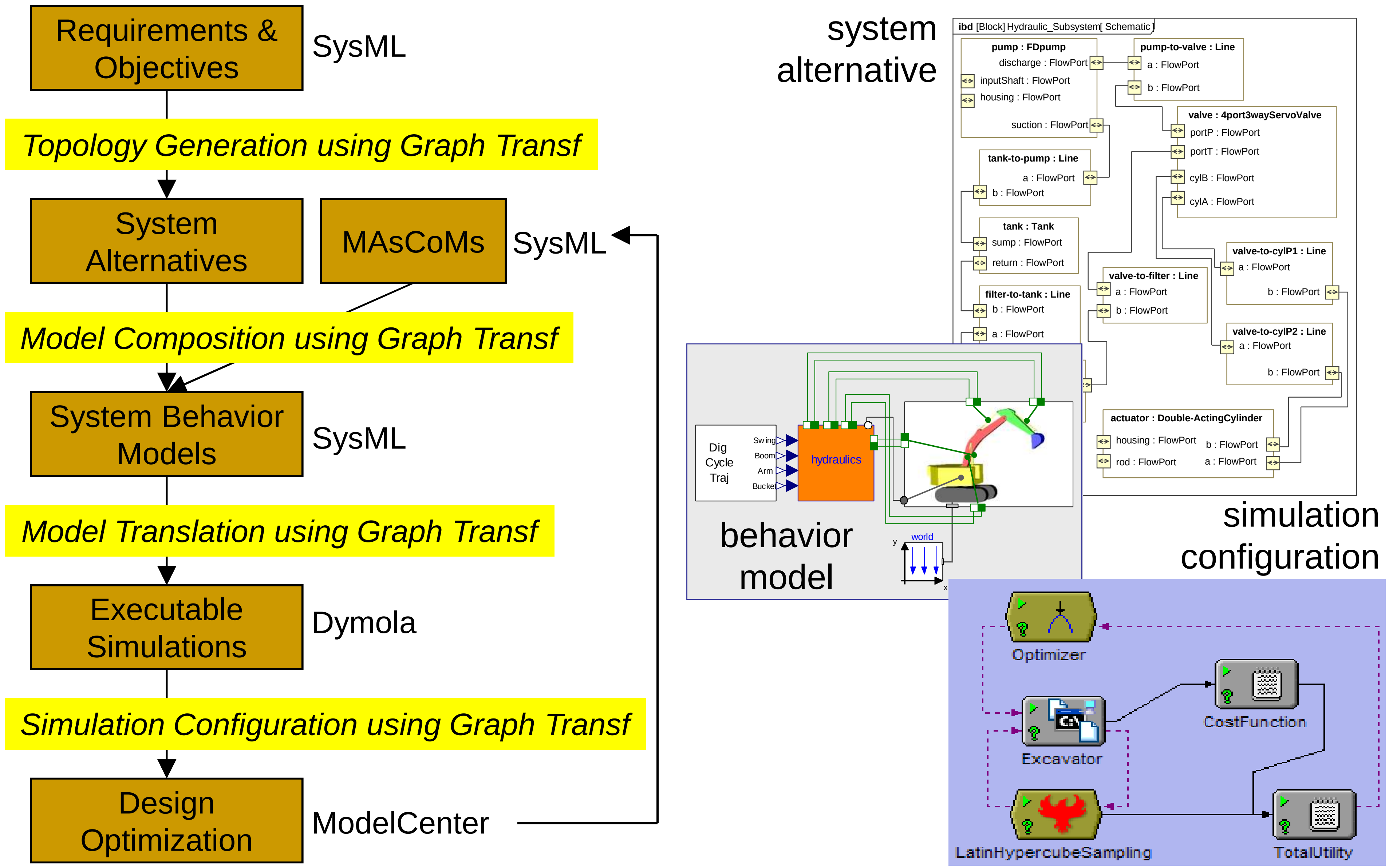


➢ Very large search and optimization problem

- Many competing objectives
- Many topologies
- Many component types/sizes
- Many control strategies

How do we best capture and use the system design knowledge?

Approach: Most Knowledge Can Be Represented as Graphs or Graph Transformations



Capturing Knowledge about Fluid-Power Circuits

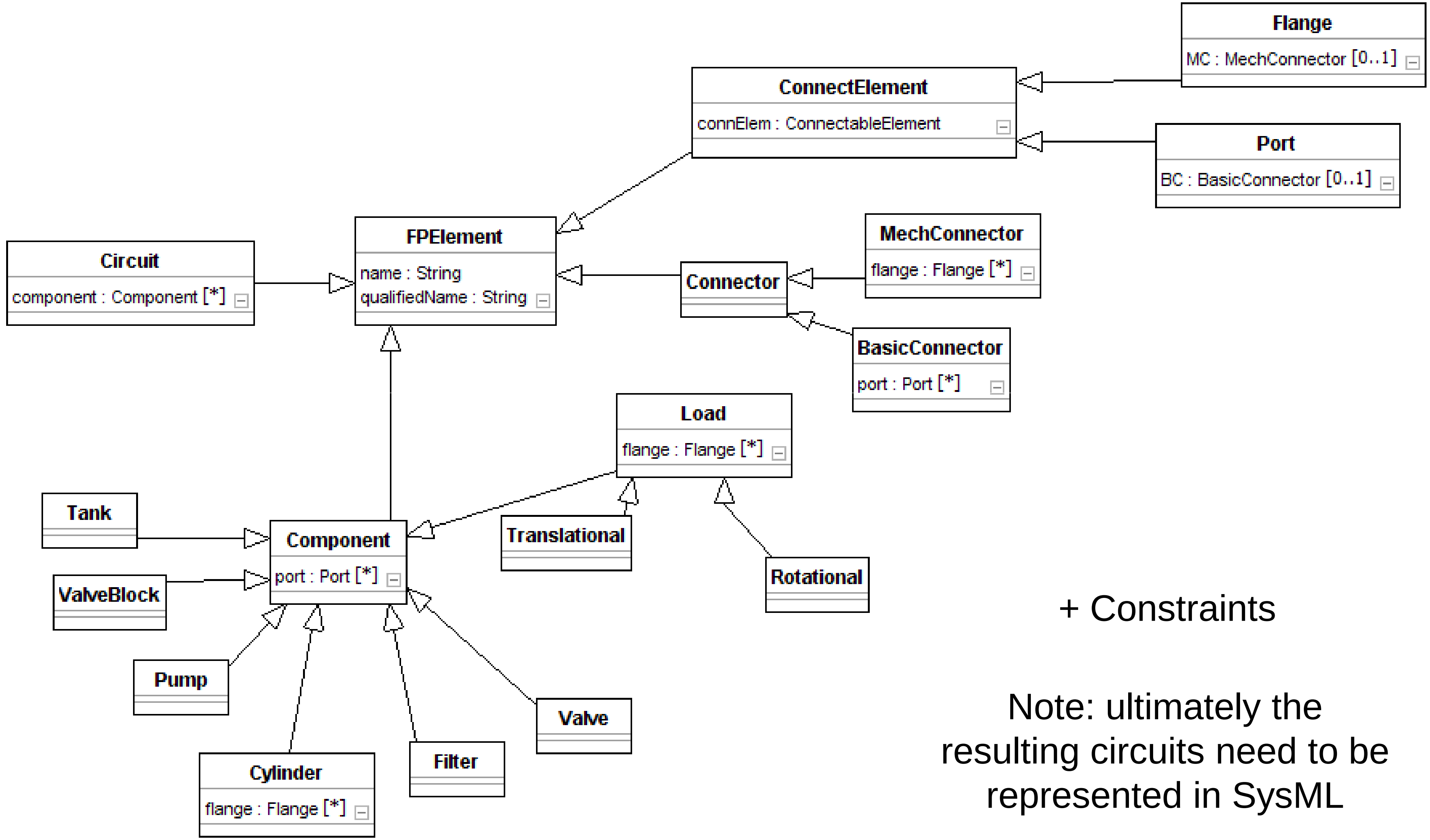
1. A Language for describing Fluid-Power circuits
   - Language is described by a meta-model
   - Valid circuits are represented as graphs

2. A Model Library with static knowledge
   - What are the available components?
   - What are their characteristics and behaviors?

3. A set of Model Transformations
   - Knowledge on how to combine components into circuit
   - Knowledge on how to generate analysis models from circuit descriptions

4. Language Mappings to/from other domains
   - Allows results to be viewed and edited (e.g. in SysML)
   - Allows circuits to be analyzed (e.g. in Dymola/Modelica)

Language for Fluid-Power Circuits

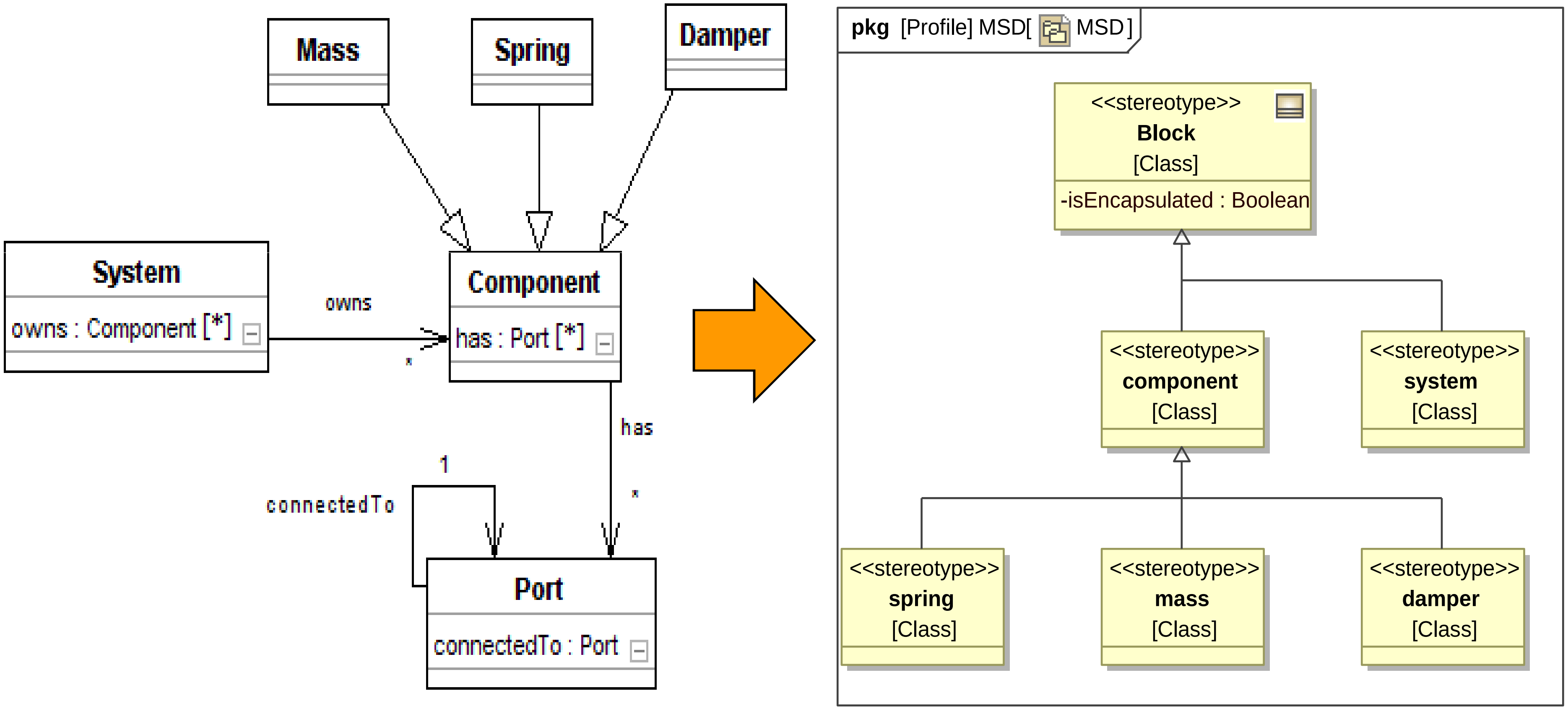


+ Constraints

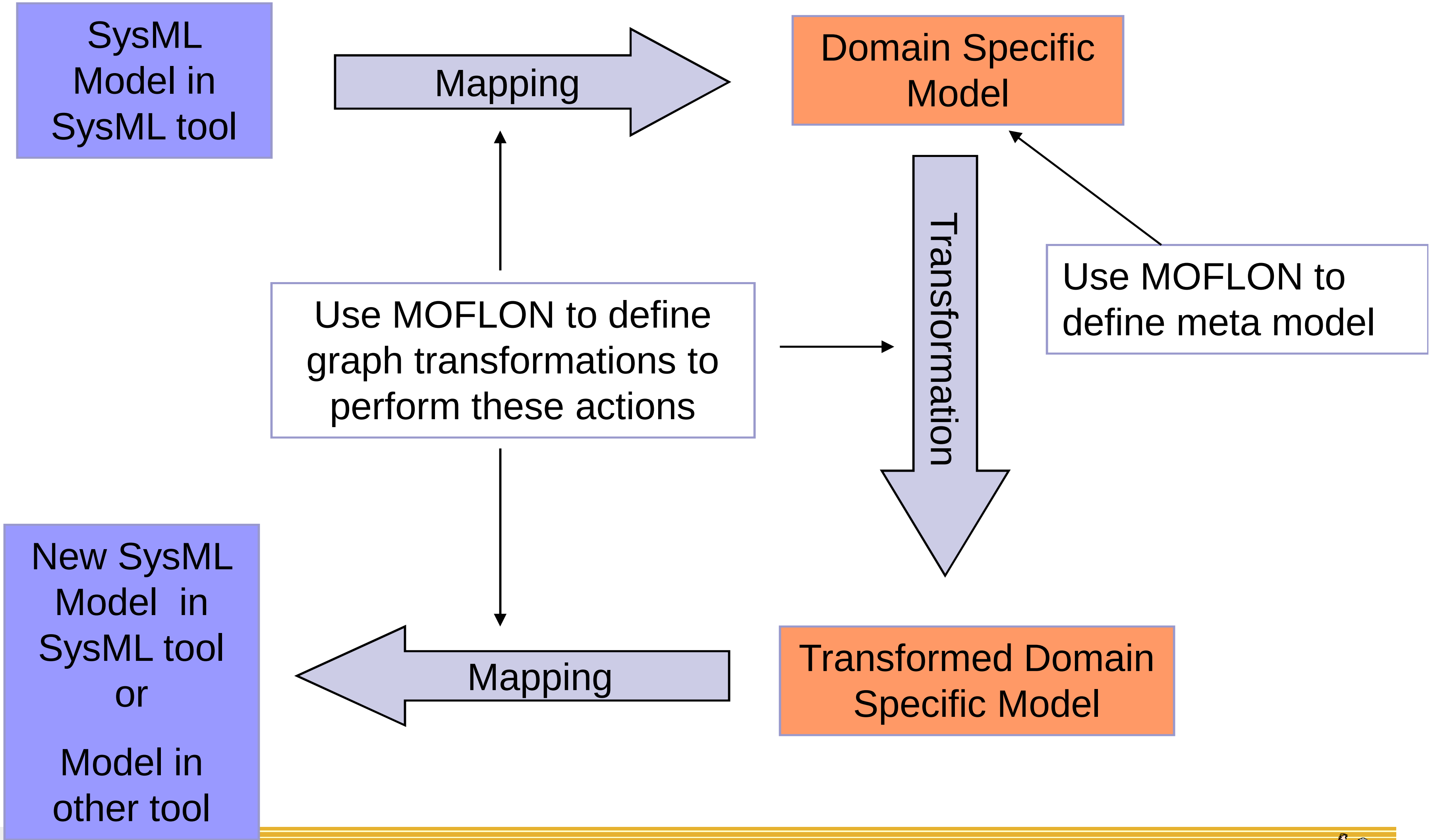
Note: ultimately the resulting circuits need to be represented in SysML

Concrete Syntax – Extending SysML

- Extend SysML using a Profile



Specify Knowledge in Domain-Specific Model



Challenges

- Language to express the Problem
  - Should cover a *set of problems* that is relevant to the user
  - The broader the set, the more complex the solution space and the more difficult the process of solving the problem could become
  - Includes objectives, requirements, etc. → very broad
  - How to anticipate all the aspects a designer may care about?

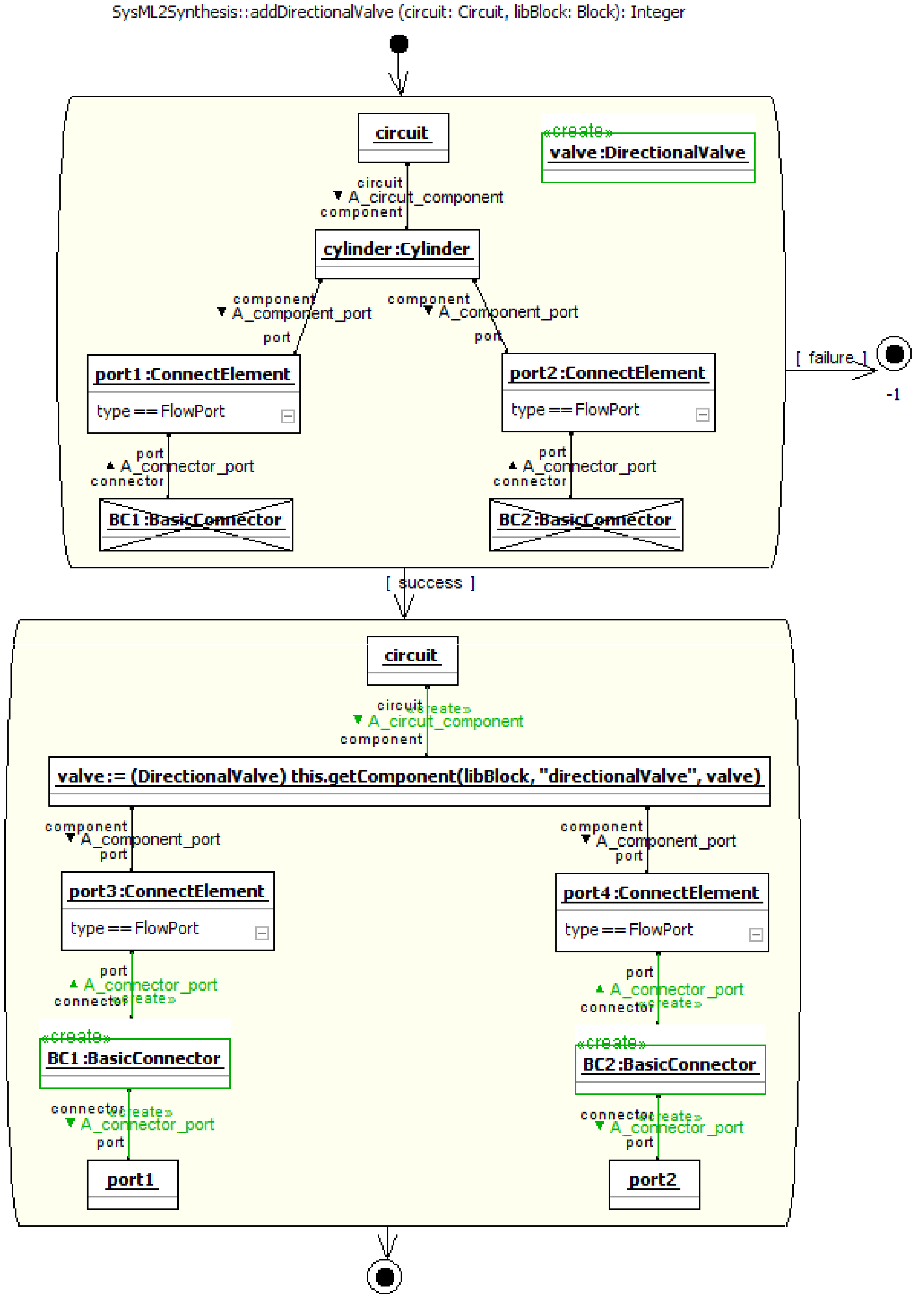
- Language to express fluid-power circuits
  - Should include each fluid-power circuit that is *optimal* for some problem instance
  - Ideally, should not include any other circuits → in practice: many more
  - Is it possible to constrain the language based on problem characteristics?

Systems Realization Laboratory

Generating System Alternatives

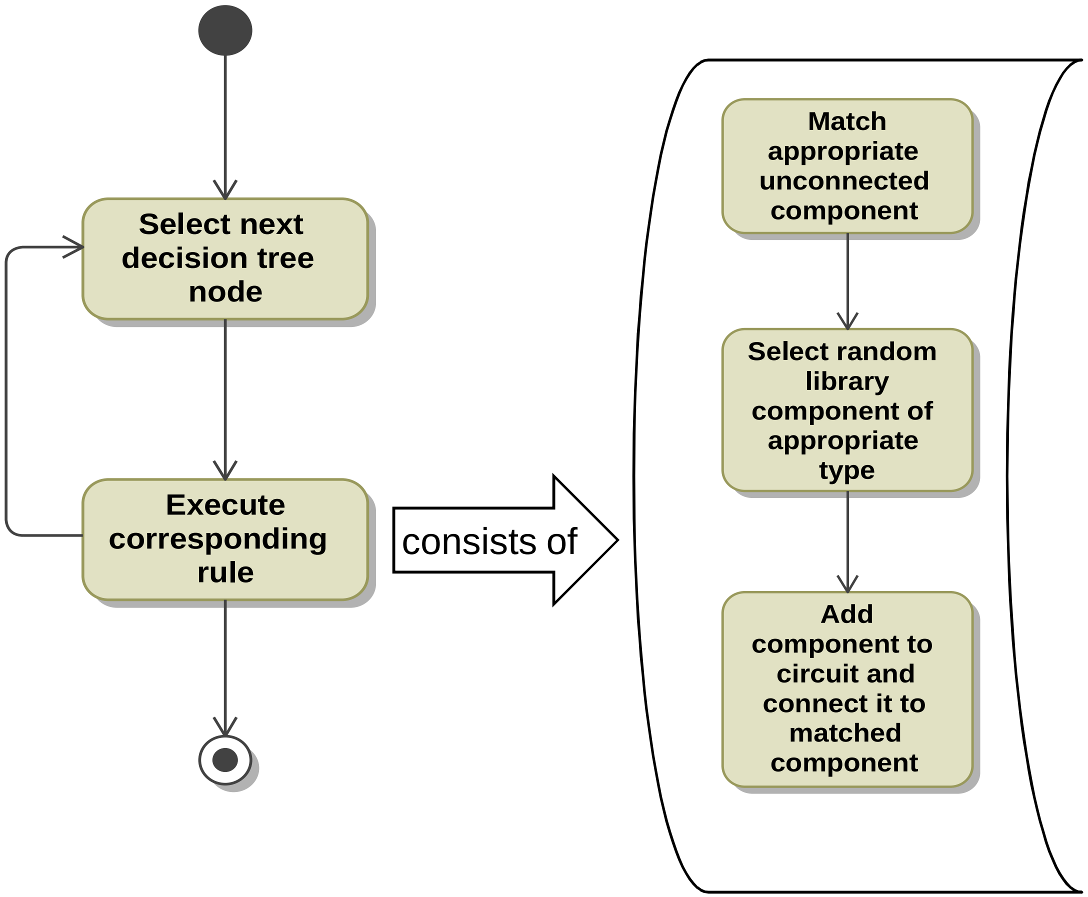
Generative Grammar for Design Synthesis

- Graph Transformation rules to generate systems
- Generate random system alternatives by applying rules in randomized order
- Improve system alternatives through evolutionary search algorithms

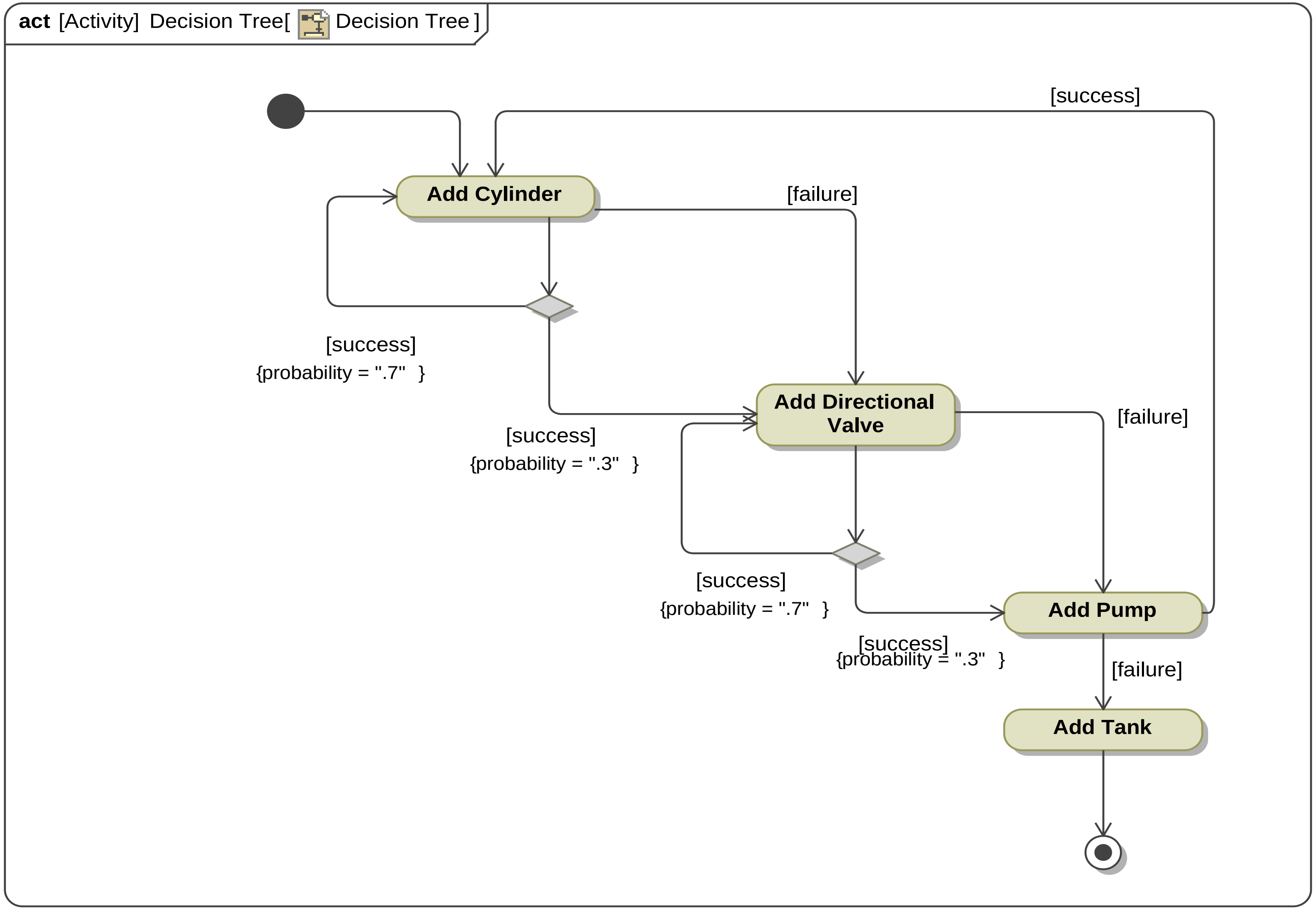


General Synthesis Approach

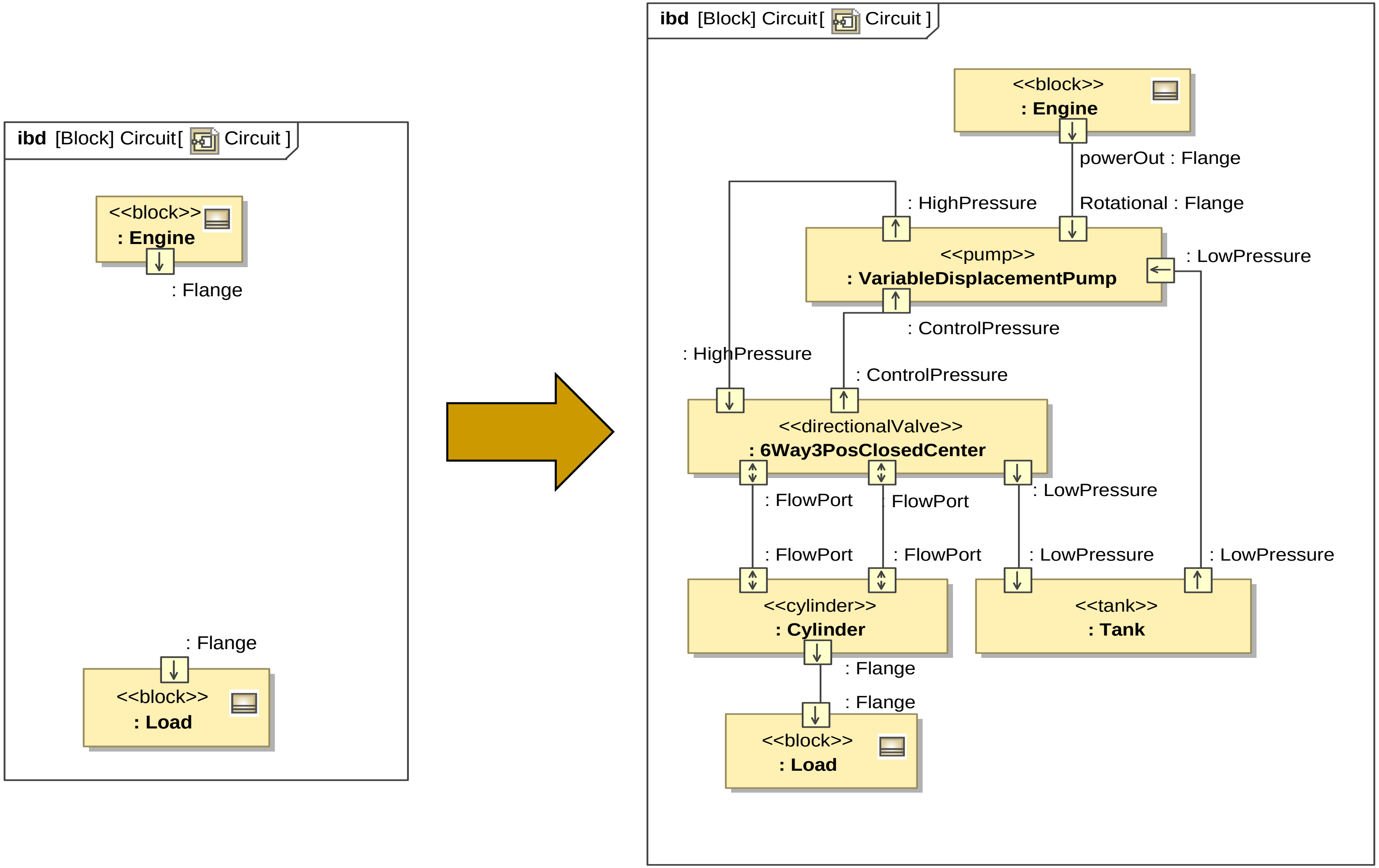
- Capture connectivity information in graph transformation rules
- Capture available components in model library
- Control the order in which rules are applied using decision tree



Decision Tree of Generation Process



Putting it all together

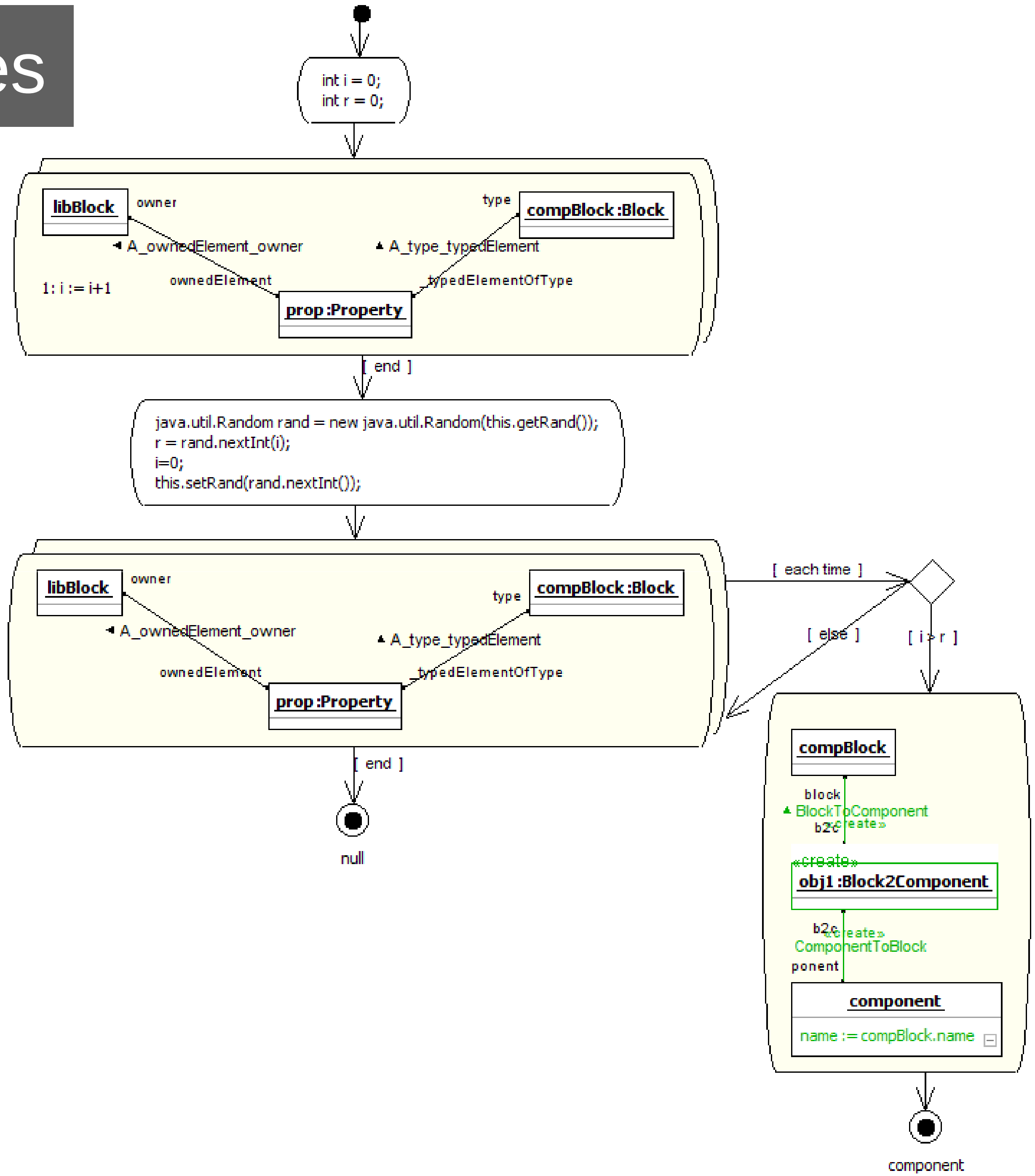


SysML2Synthesis::getComponent (libBlock: Block, stereotype: String, component: Component, previousComponent: Component): Component

Some Challenges

- Selecting components at random:
  - Instead of simply matching one instance, need to match one instance *at random*

- Rule set
  - Should cover the entire space of circuits
  - Randomness should be "uniform" across space



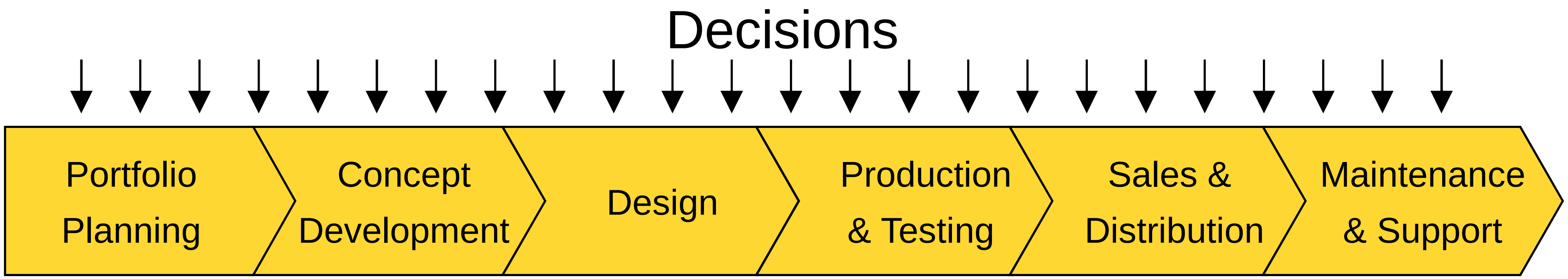
Challenges

- How to impose constraints in a generative grammar?
  - We have only explored graph transformations…
  - Could we accomplish the same using constraint-based meta-model defined in Alloy (or similar tool)?
  - Which approach is most intuitive/convenient for domain experts?
  - Which approach is best suited for automated (randomized) synthesis and incremental modification (as in optimization/search)?
- Larger problems:
  - Which knowledge is captured in synthesis model and which is left for analysis?
  - How to work at different levels of abstraction?
    - E.g.: topology, sizing, control,…
  - Is there a systematic process for capturing synthesis knowledge?

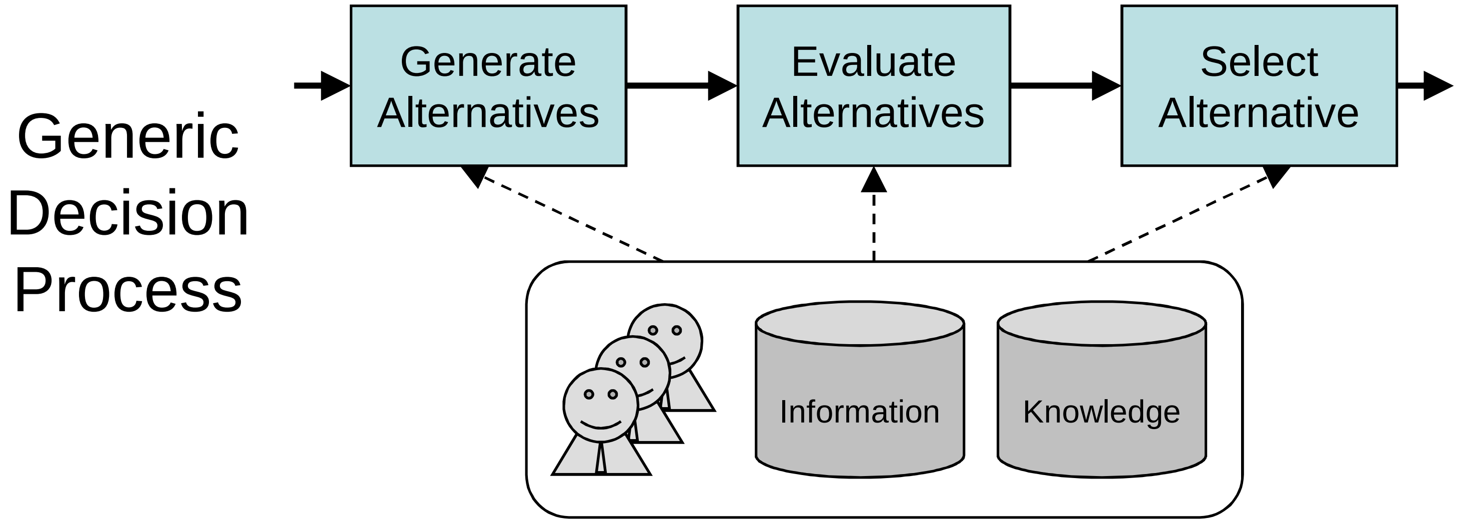
Systems **R**ealization **L**aboratory

Generating System-Level Analysis Models

Systems Development: A Decision-Based Perspective

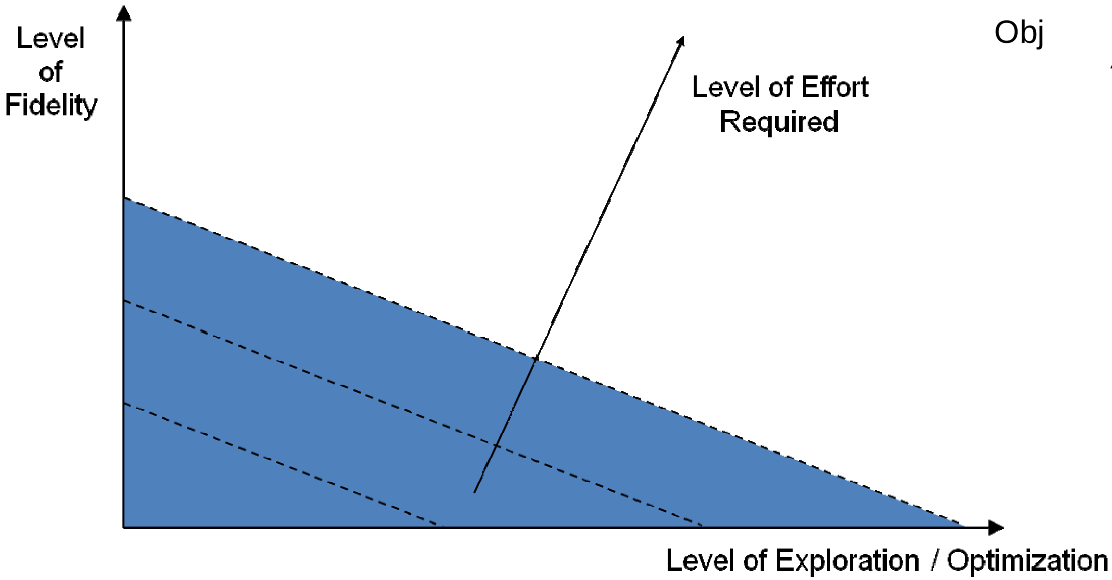


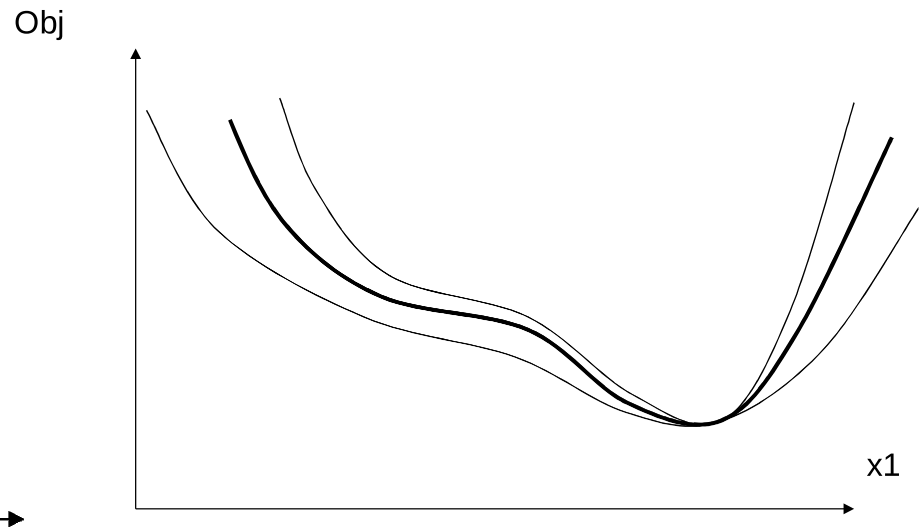
Modeling and Simulation Provides Information in Support of Decisions



Challenges

- Many different perspectives, levels of abstraction, formalisms
- Hypothesis:
  - One can improve the efficiency of design optimization methods by considering multiple levels of abstraction and accuracy





Objective Function and its Accuracy Bounds

Model Management Problem

- # models = O(#system topologies) *
  O(#attributes) *
  O(#abstraction levels) *
  O(#fidelities)

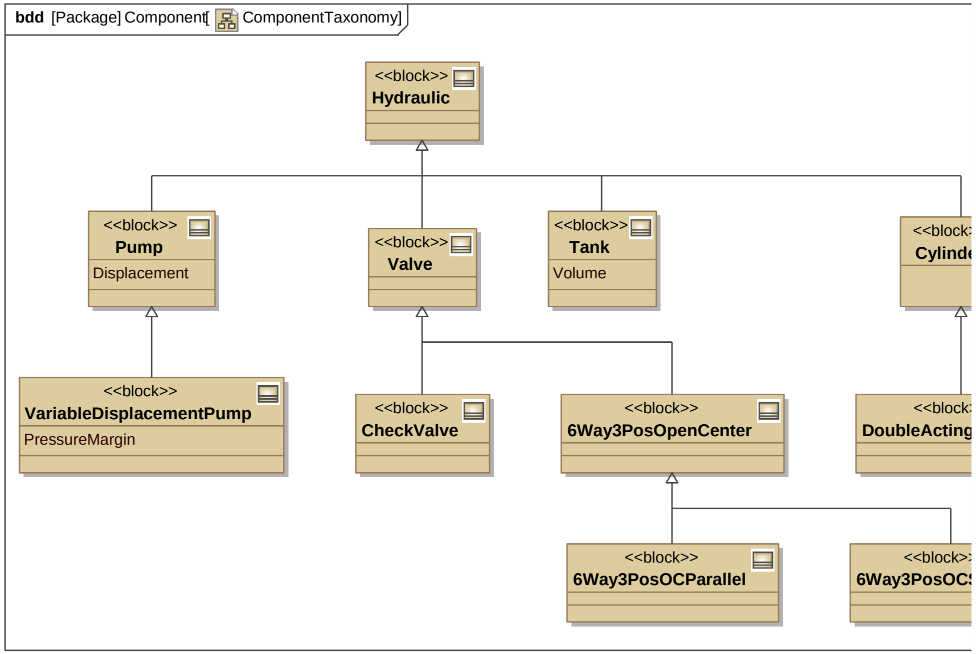
- How do we manage all these models?

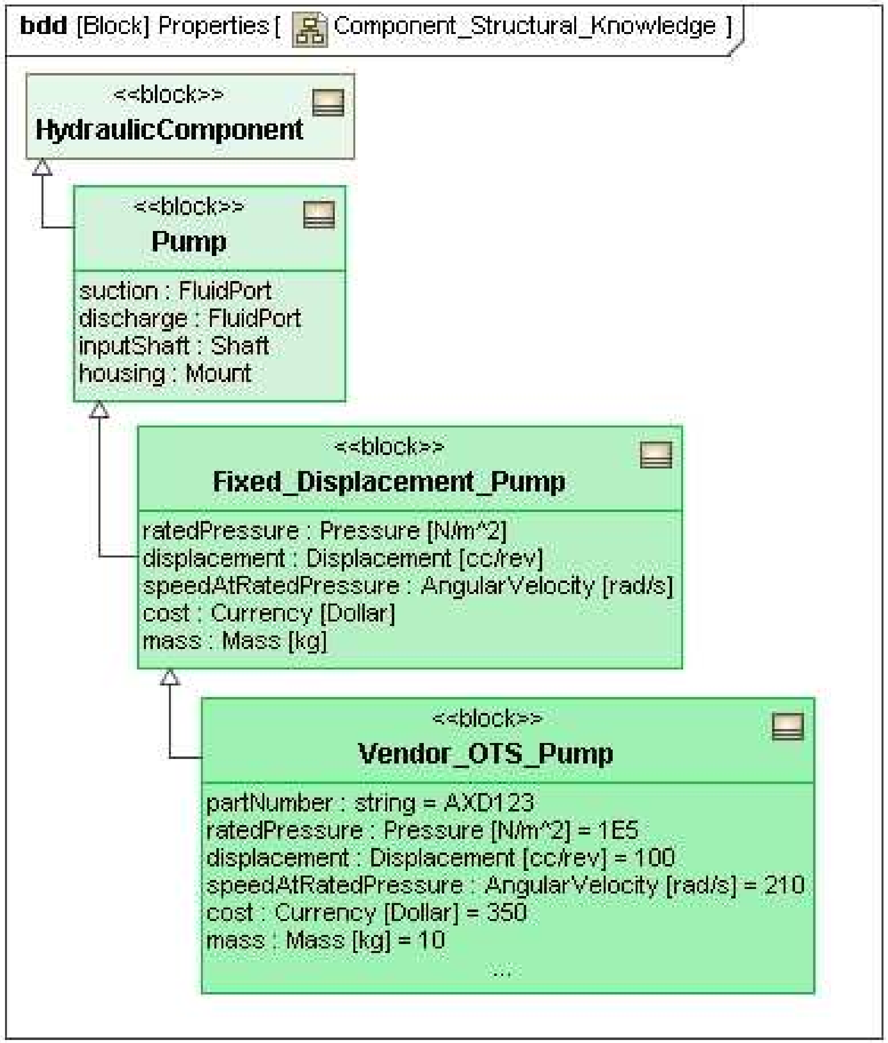
  → Use model transformations to generate the models as needed

1. Create *specific* transformation rules to generate analysis models
2. Create *general* rules for composition based on model correspondence templates in library

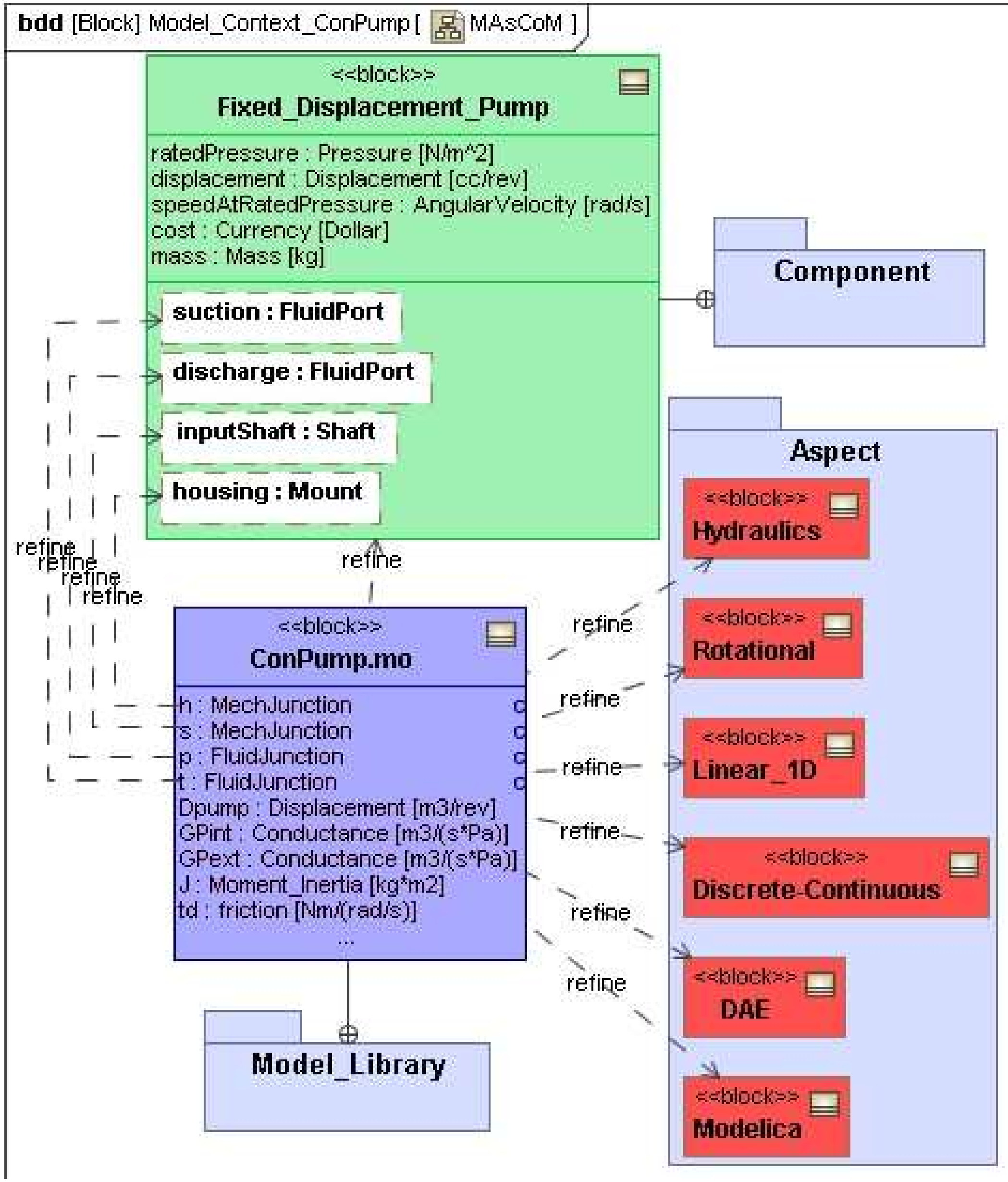
Library of Fluid Power Components

- Vocabulary of the synthesis grammar





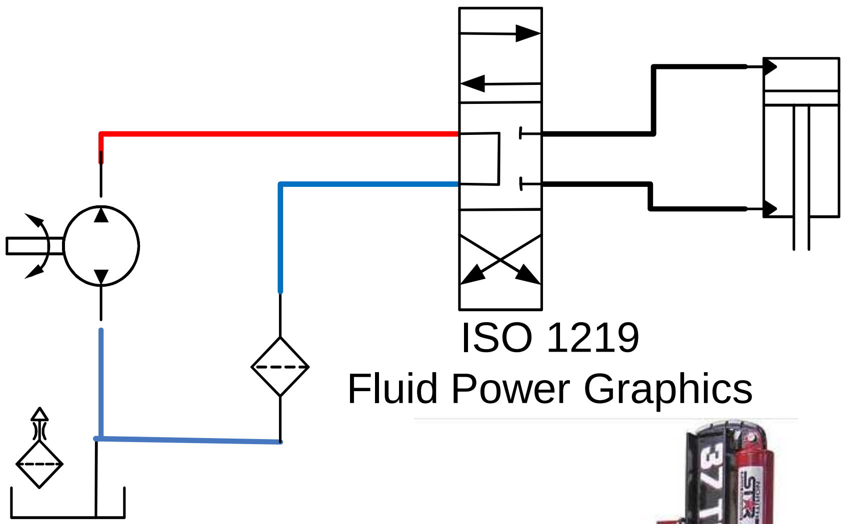
Model Library of Fluid Power Components



- Library of Fluid Power components
  - Defined as MAsCoMs (Multi-Aspect Component Models)
- Components are the reusable elements of design
- Multi-Aspect Component Models (MAsCoMs):
  - Group all models related to single fluid power component
  - Multiple disciplines and levels of abstraction
  - Modular
  - Formal & unambiguous

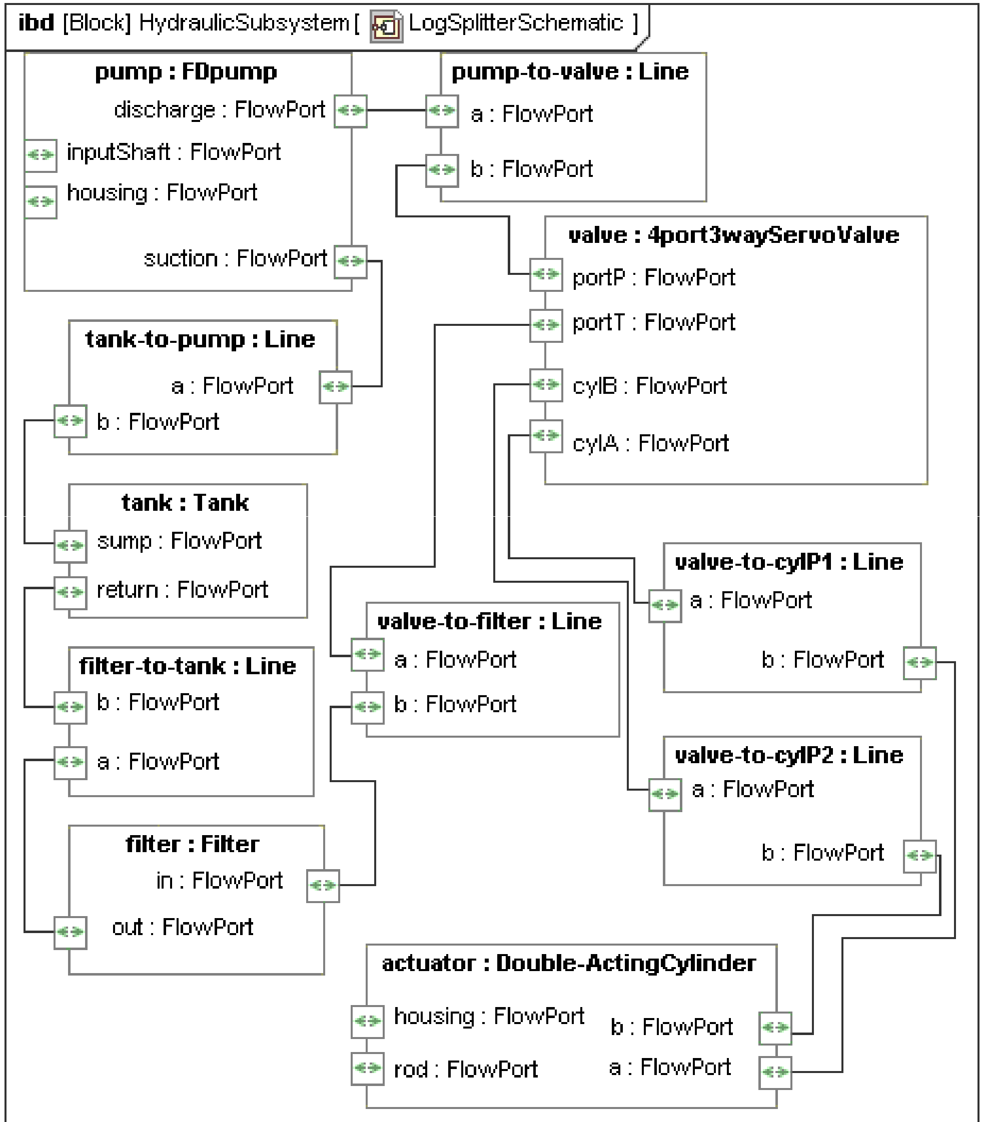
How to use MAsCoMs?

Log Splitter Design Example

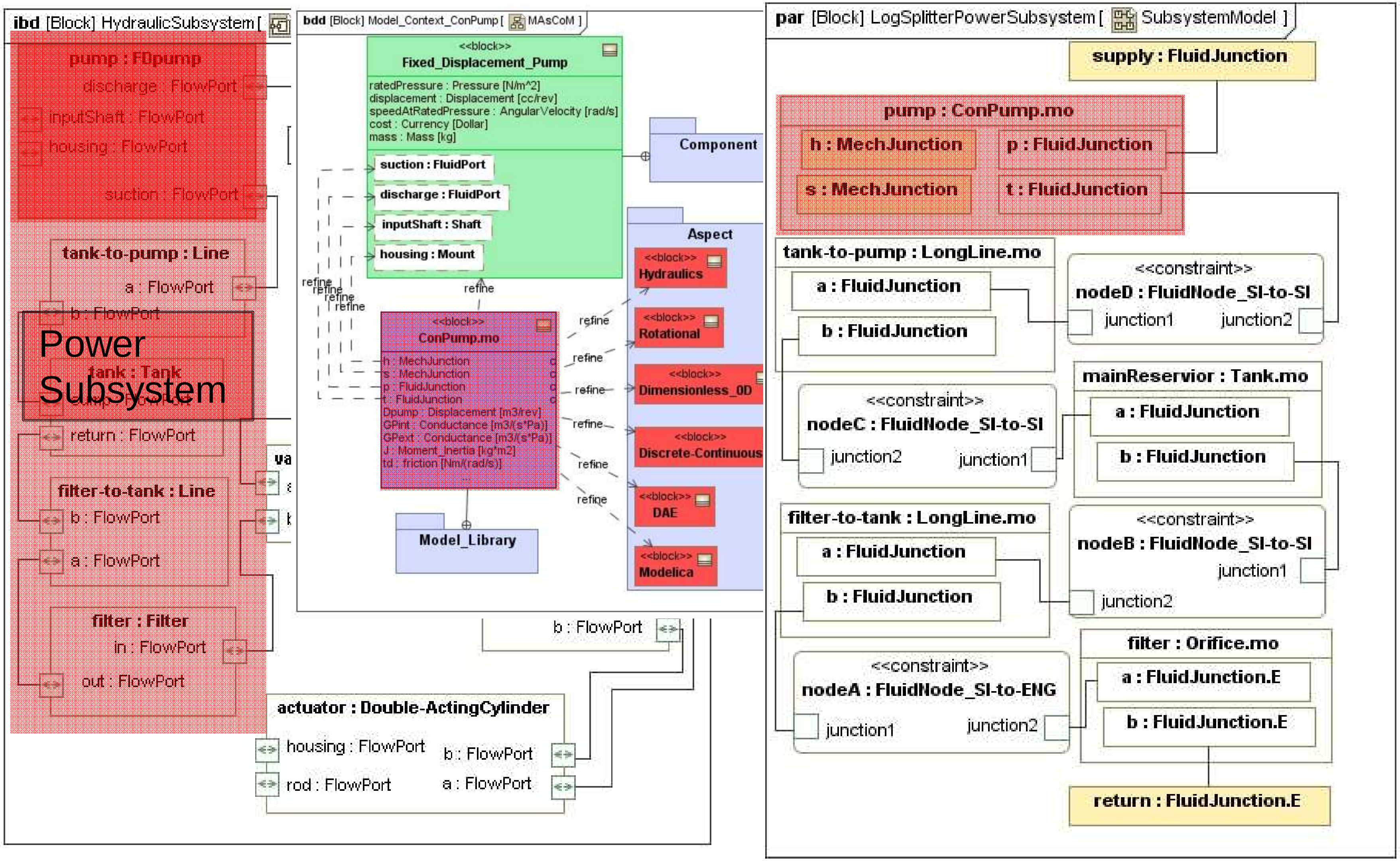


ISO 1219
Fluid Power Graphics

Design Concept Schema
-Hydraulic System



Composition of Correspondence Templates



Generating System-Level Analyses

- Principle: Separation of Viewpoints
  - Separate model for each analysis perspective
  - Don't mix analysis and structure models
- Approach: Composition
  - Compose component models into system-level model
  - Encode the composition rules as model transformations
  - Organize the composition patterns in a model library
  - Different types of models require different composition rules

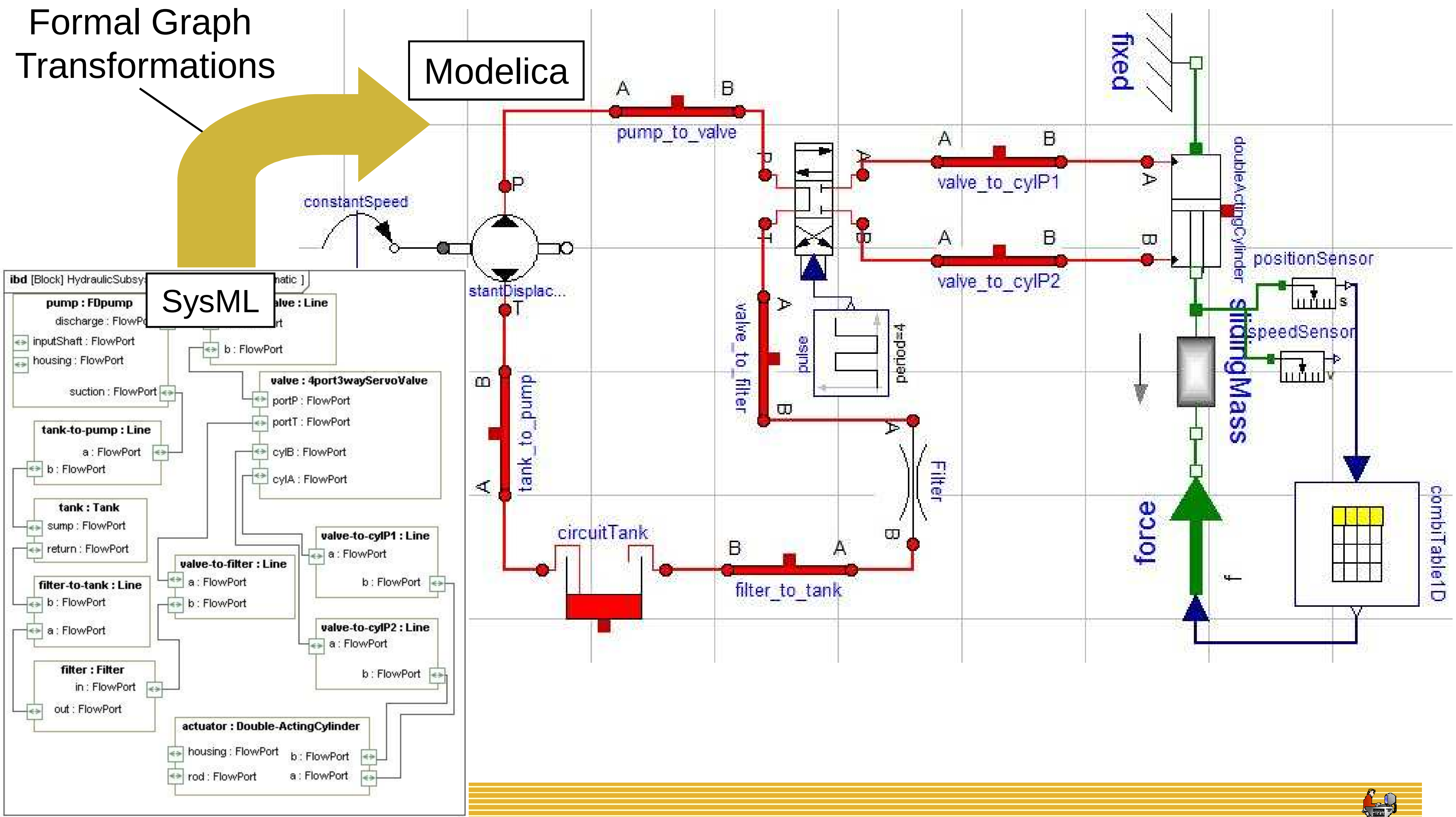
Challenges

- How to select the “right” component-level models?
  - Perspective, compatibility, accuracy,…
  - Cost-benefit trade-off requires meta-information about models
    - Cost, accuracy, applicability,…
- What are the different composition transformations?
  - Transformation depends on formalism
- What happens if the composition transformation requires additional information?
  - E.g., synthesize structural description → convert to behavior → not all physical behavior parameters are available

Model Mapping

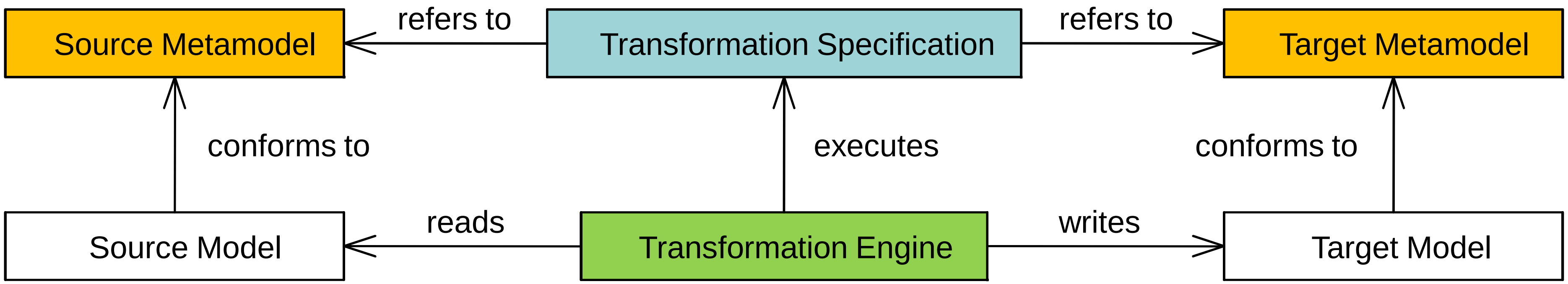
Systems **R**ealization **L**aboratory

Automatic Translation from SysML to Modelica



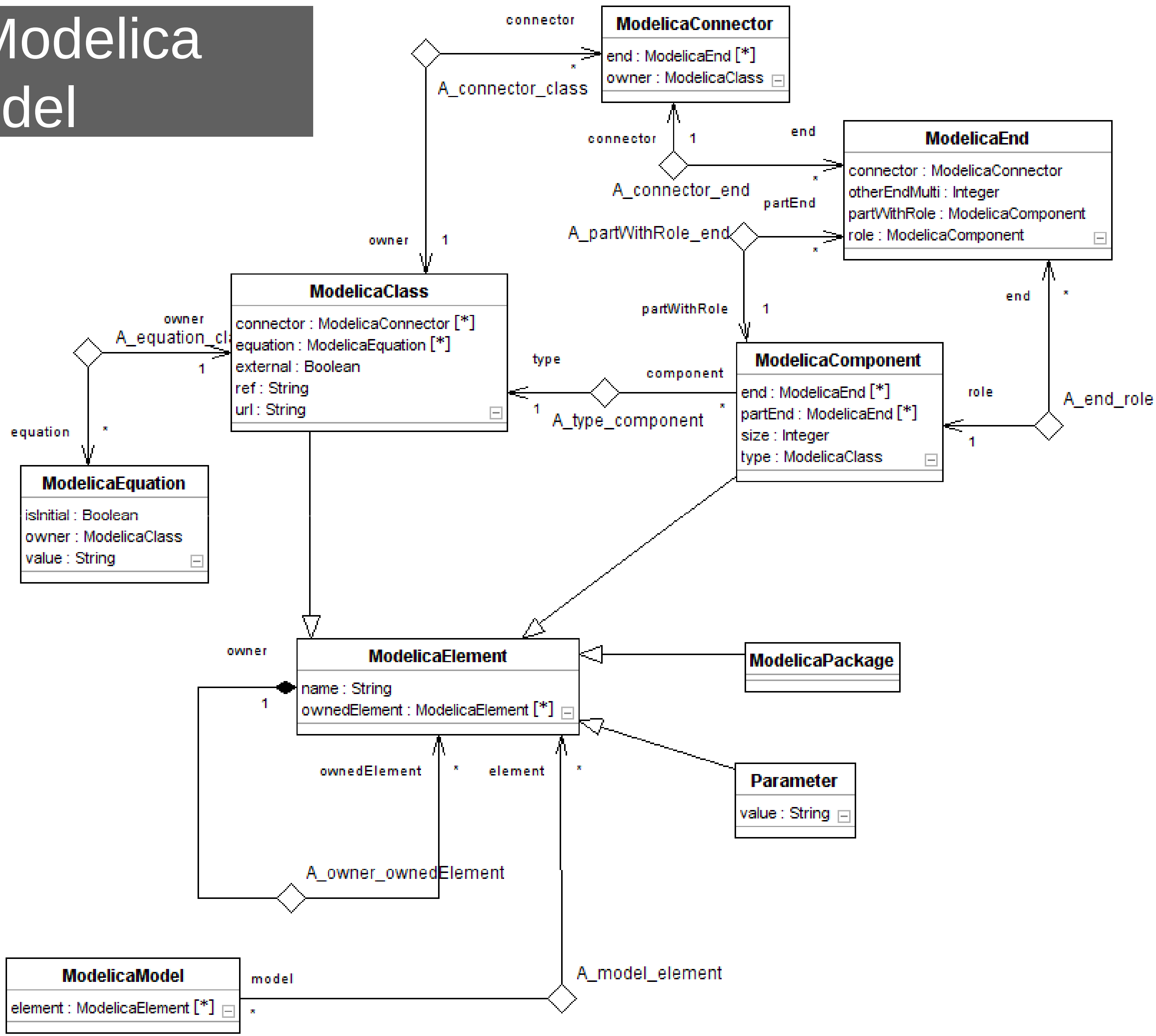
Mapping between SysML and Other Languages
(based on work by Andy Schürr)

1. Define meta-models
   - May require reverse-engineering meta-model
2. Create JMI adapter for tools
3. Define a model transformation
   - Create graphs of correspondence between meta-models
   - Triple Graph Grammar (TGG)
4. Compile rules (MOFLON) and load as plug-in

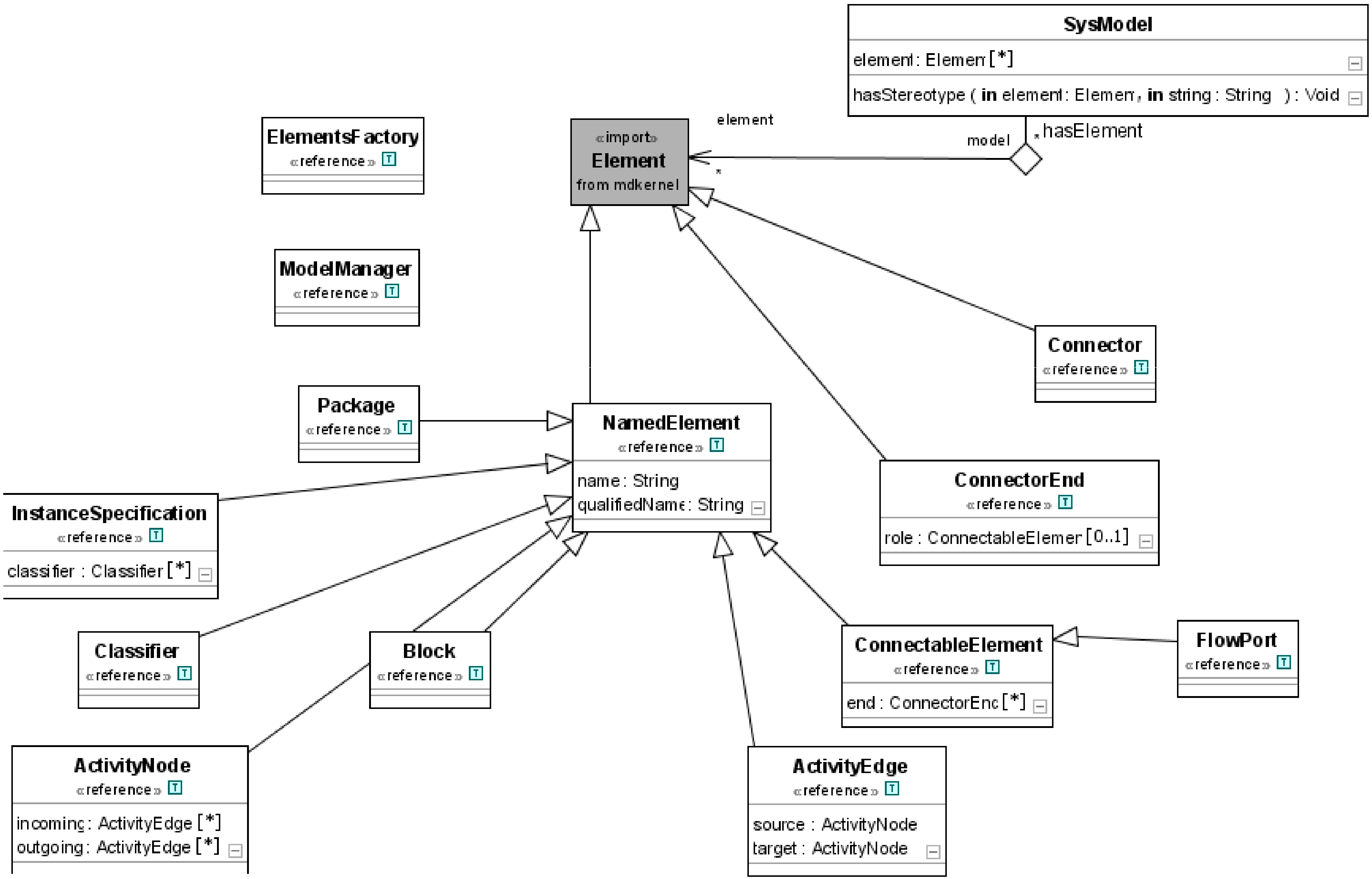


(Czarnecki, K., & Hellen, S., 2006)

Partial Modelica Metamodel

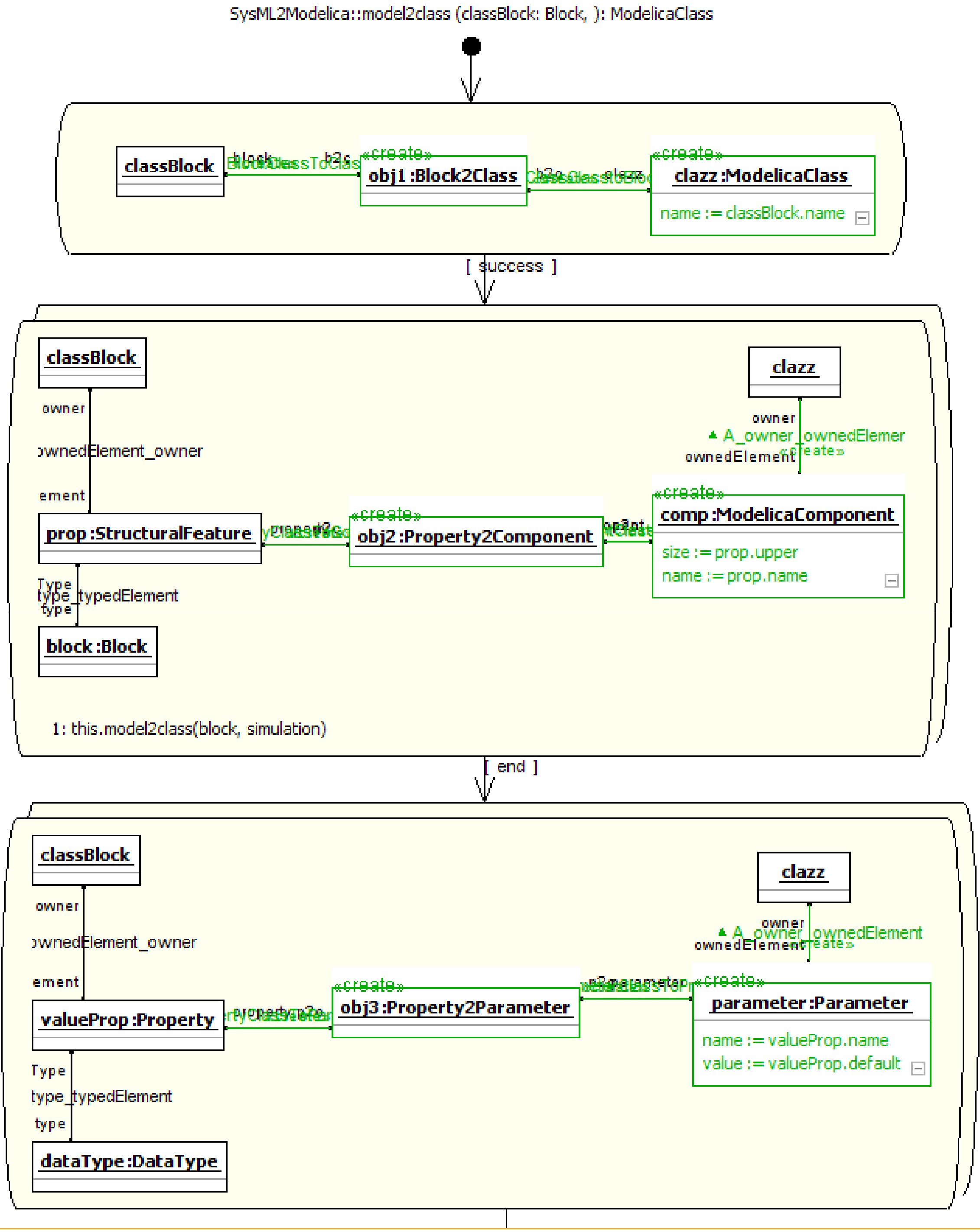


Partial SysML Metamodel in MOFLON

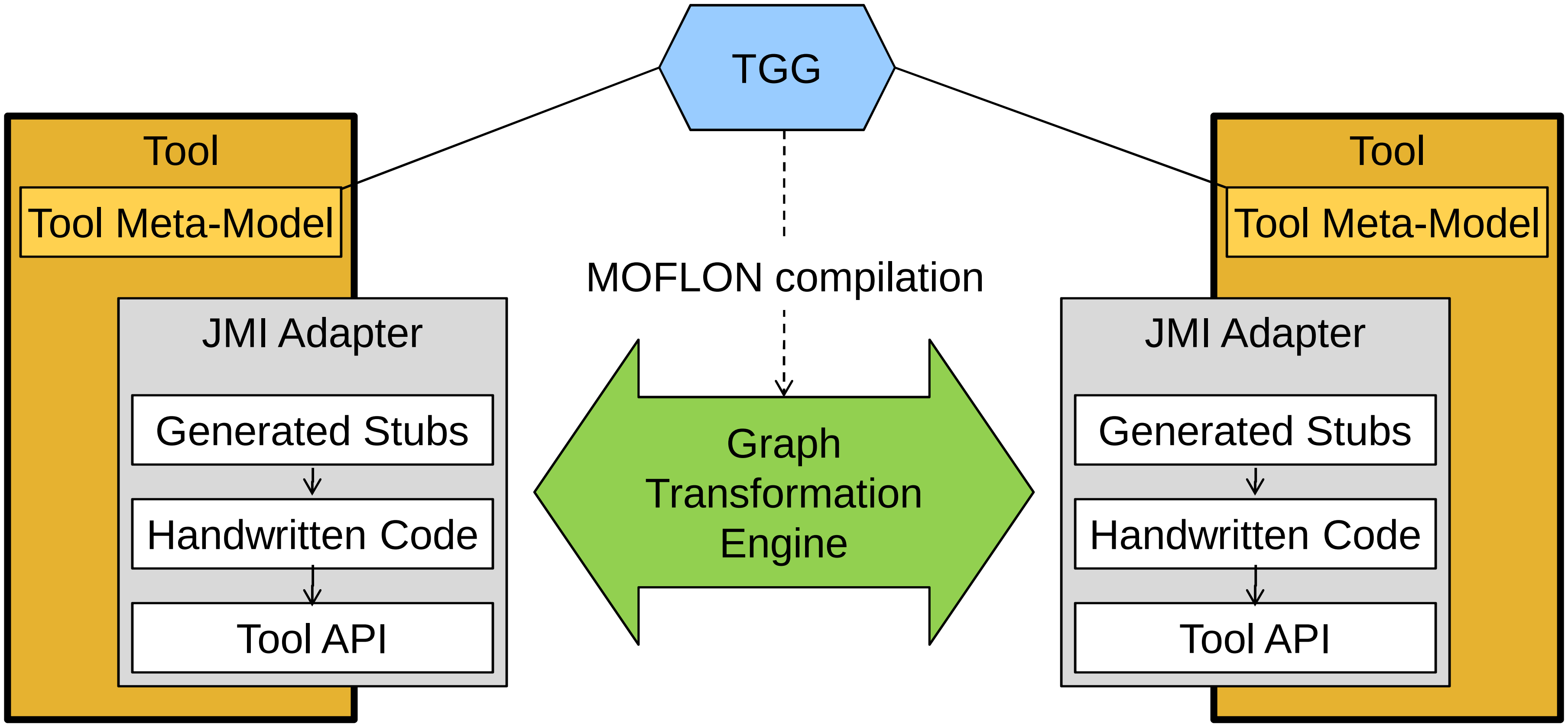


Transformation Rules

- Could be automatically generated through Triple Graph Grammar mechanism



TGG Mapping Mechanism in MOFLON



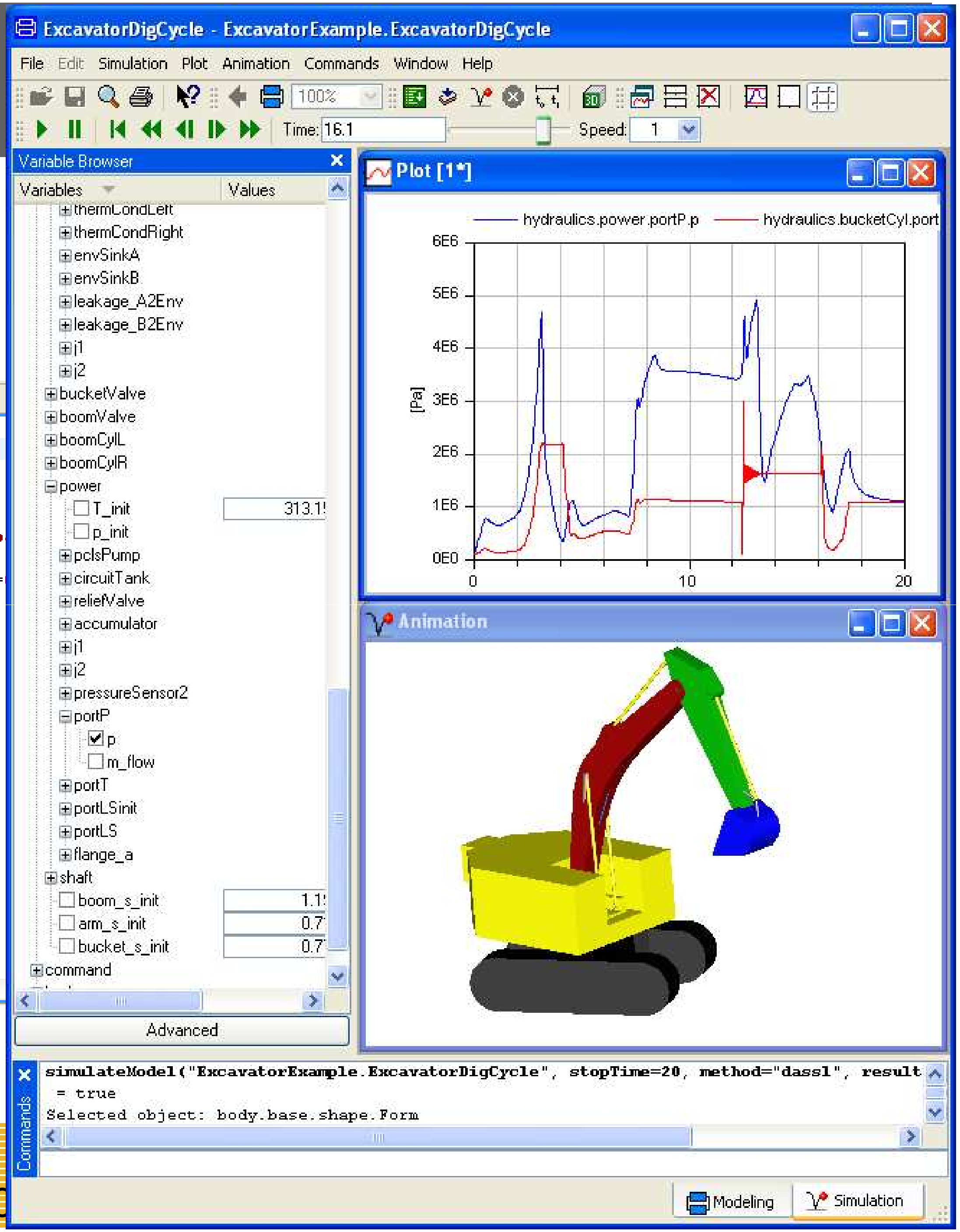
(Note: My interpretation of work by Andy Schürr)

Simulation in Dymola

Modelica Lexical Representation (auto-generated from SysML)

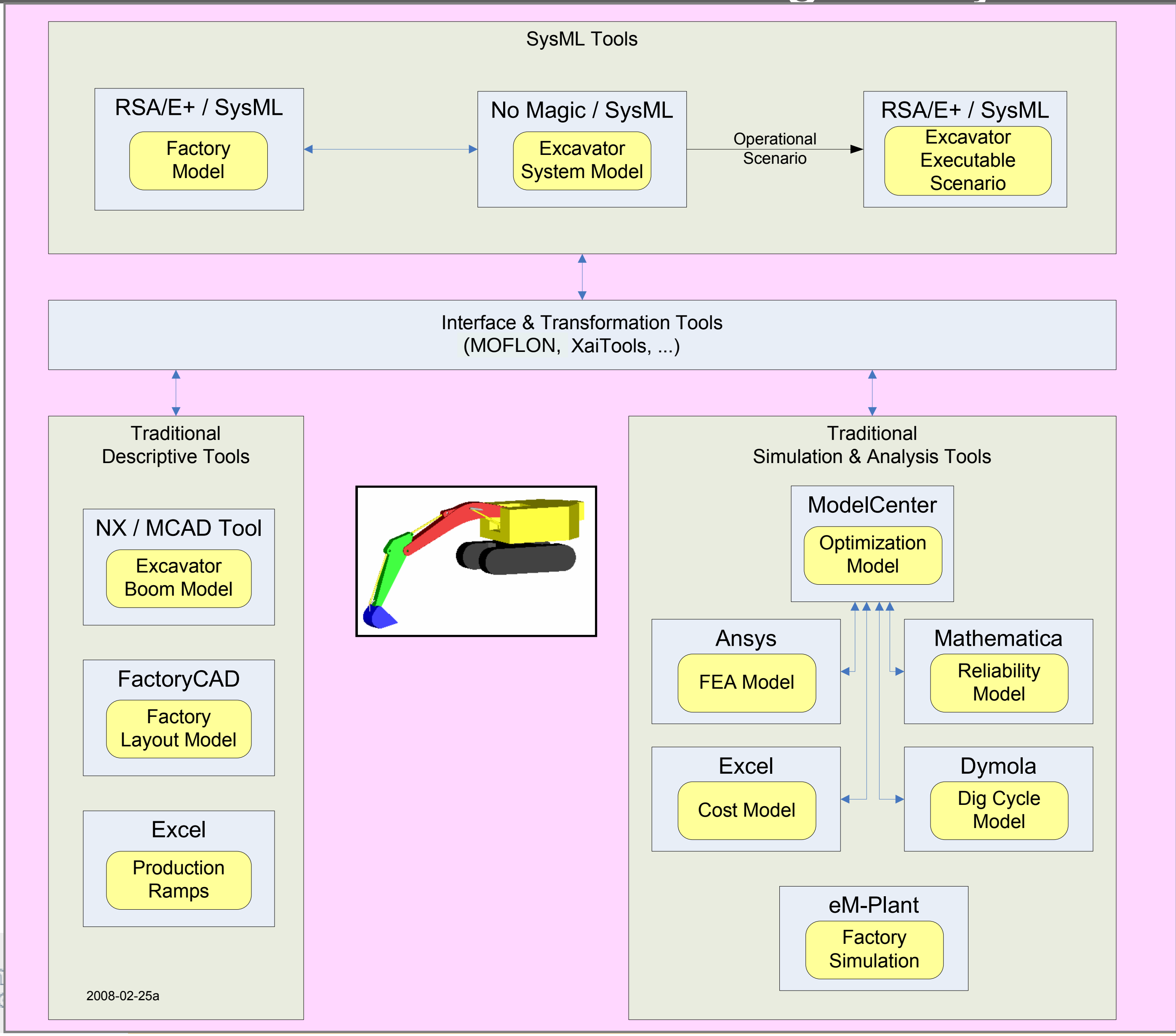
Simulation Results

```
*ExcavatorExample.mo
package ExcavatorExample
  ...

  class ExcavatorDigCycle
    Modelica.Mechanics.MultiBody.World world;
    ExcavatorExample.Components.Hydraulics hydraulics(redeclare p
    ExcavatorModel.SubSystems.DigCycleSeq command(startTime=0.1);
    ExcavatorModel.SubSystems.MechanicsBody body(swing_phi_start=
    ExcavatorExample.Interfaces.Nodes.TransNode2 node;
  equation
    connect(hydraulics.boomCylBaseL, body.cylBoomLeftBase);
    connect(hydraulics.boomCylRodR, body.cylBoomRightRod);
    connect(hydraulics.boomCylRodL, body.cylBoomLeftRod);
    connect(hydraulics.armCylRod, body.cylArmRod);
    connect(hydraulics.armCylBase, body.cylArmBase);
    connect(hydraulics.bucketCylRod, body.cylBucketRod);
    connect(hydraulics.bucketCylBase, body.cylBucketBase);
    connect(hydraulics.commandSignal, command.commandSignal);
    connect(world.frame_b, body.baseFrame);
    connect(hydraulics.swingFlange, body.swingFlange);
    connect(hydraulics.boomCylBaseR, node.a);
    connect(node.b, body.cylBoomRightBase);
  end ExcavatorDigCycle;

end ExcavatorExample;
```



[Johnson, 2008 - Masters Thesis]

SysML Tool-Integration: INCOSE MBSE Challenge Project



Challenges

- How general is the TGG approach?
  - Is there a point at which it breaks down?
  - Limitations of bidirectional mappings?
- Is there a universal way to interface with disciplinary tools?
  - Is a JMI adapter the best way?
- And here I ran out of time… ☺

  … time to summarize

Summary of Approach

1. A Language for describing Fluid-Power circuits
    - Language is described by a meta-model
    - Valid circuits are represented as graphs
2. A Model Library with static knowledge
    - What are the available components?
    - What are their characteristics and behaviors?
3. A set of Model Transformations
    - Knowledge on how to combine components into circuit
    - Knowledge on how to generate analysis models from circuit descriptions
4. Language Mappings to/from other domains
    - Allows results to be viewed and edited (e.g. in SysML)
    - Allows circuits to be analyzed (e.g. in Dymola/Modelica)

Acknowledgements

- Sponsors
  - National Science Foundation: Center for Compact and Efficient Fluid Power
  - Deere & Co
  - Lockheed Martin
- Collaborators
  - Roger Burkhart
  - Sandy Friedenthal
  - Leon McGinnis
  - Russell Peak
  - Dirk Schaefer
- Students
  - Jonathan Bankston
  - Jonathan Jobe (graduated)
  - Tommy Johnson (graduated)
  - Alek Kerzhner
  - Aditya Shah

Questions?