

CSSE 490 Model-Based Software Engineering: Software Factories



Shawn Bohner

Office: Moench Room F212

Phone: (812) 877-8685

Email: bohner@rose-hulman.edu

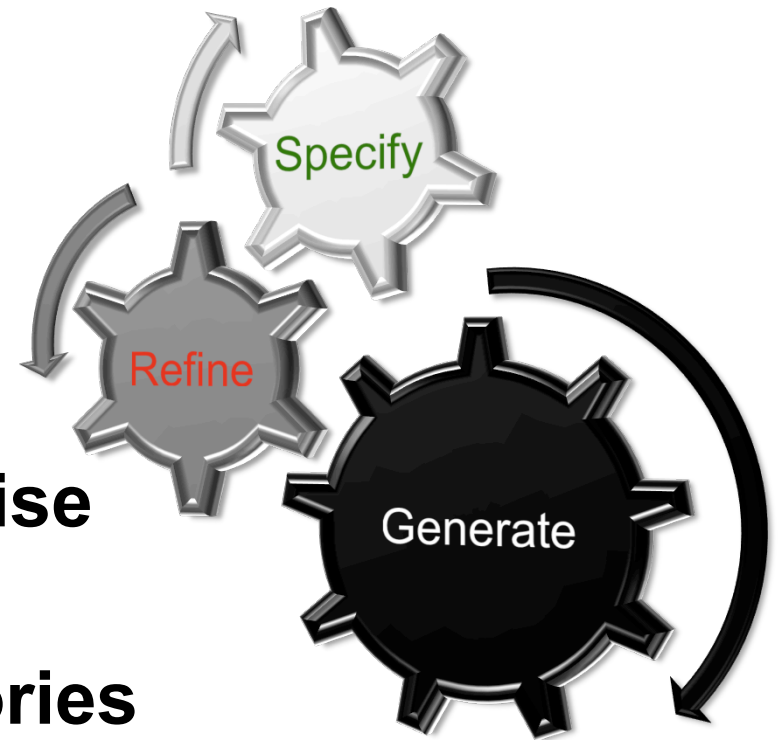


ROSE-HULMAN
INSTITUTE OF TECHNOLOGY

Learning Outcomes: MBE Discipline

Relate Model-Based Engineering as an engineering discipline.

- Discussion of Milestone 3
- Finish Class Project Exercise
- Example: DRACO
- Context for Software Factories
- Key Software Factory Concepts
- Short Microsoft SF Example



Exercise: Reflect on Class Project

Describe how Milestone 3 Environment Automates Software Development.

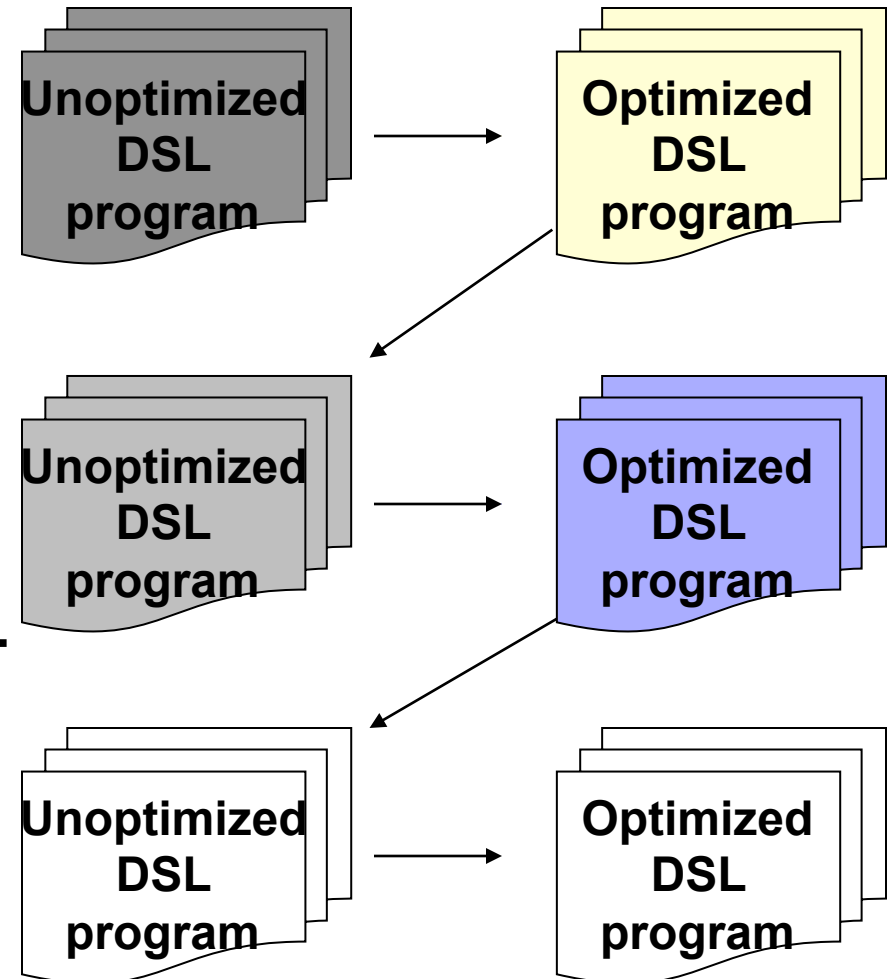
- What are the assembling steps?
- What are the transformational steps?
- Is the domain language a separate concern?
- Draw a diagram of how it works?



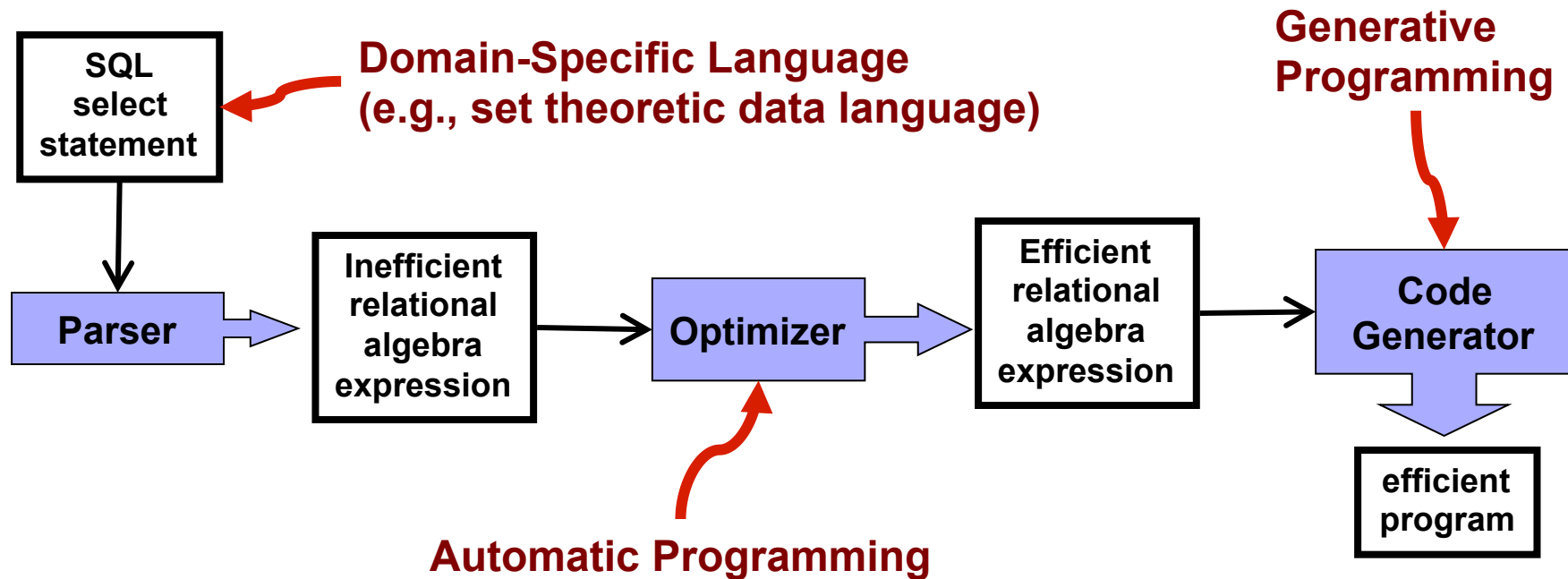
Early Generative Reuse System

DRACO – Jim Neighbors (1984)

- Programs are written in domain-specific languages
- Optimize DSL programs (because the domain abstractions are visible)
- Translate DSL to another, lower-level abstraction DSL and do the same
- Process repeats until you get to machine code

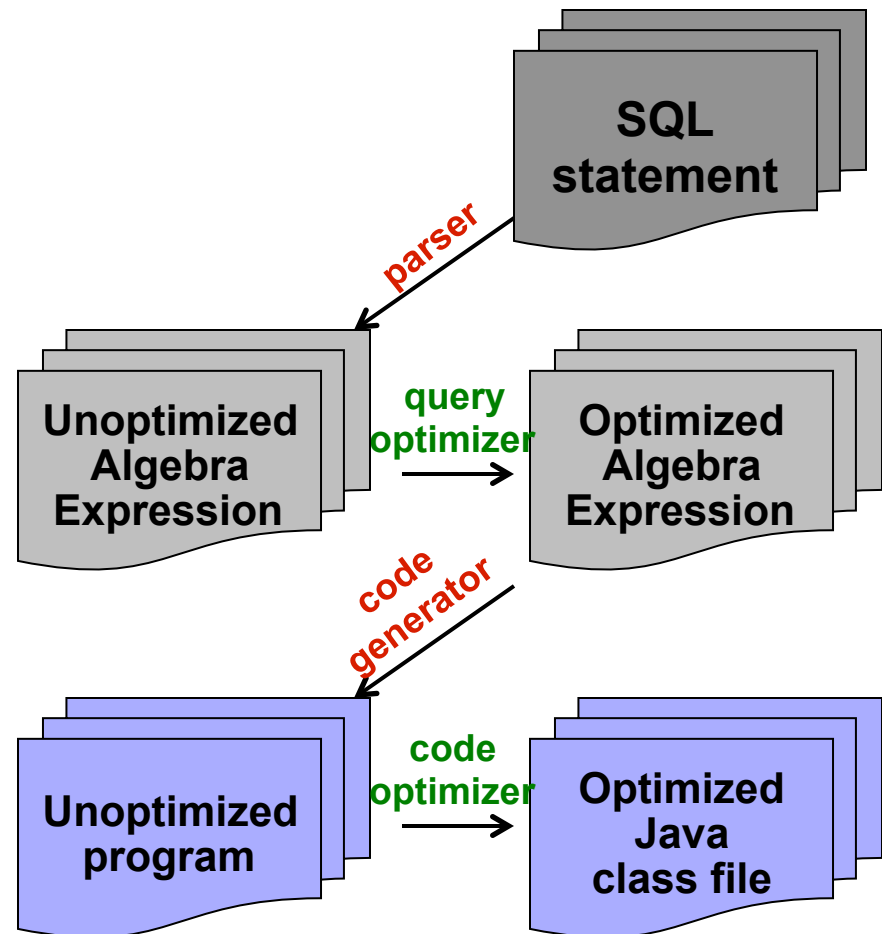


DRACO Perspective of Programming



DRACO View of Query Optimization

- **Generative Programming** occurs when mapping between levels of abstraction
- **Automatic Programming** occurs when optimizing (or refining/transforming) within a level of abstraction is done automatically





MBSE Requires Advances in:

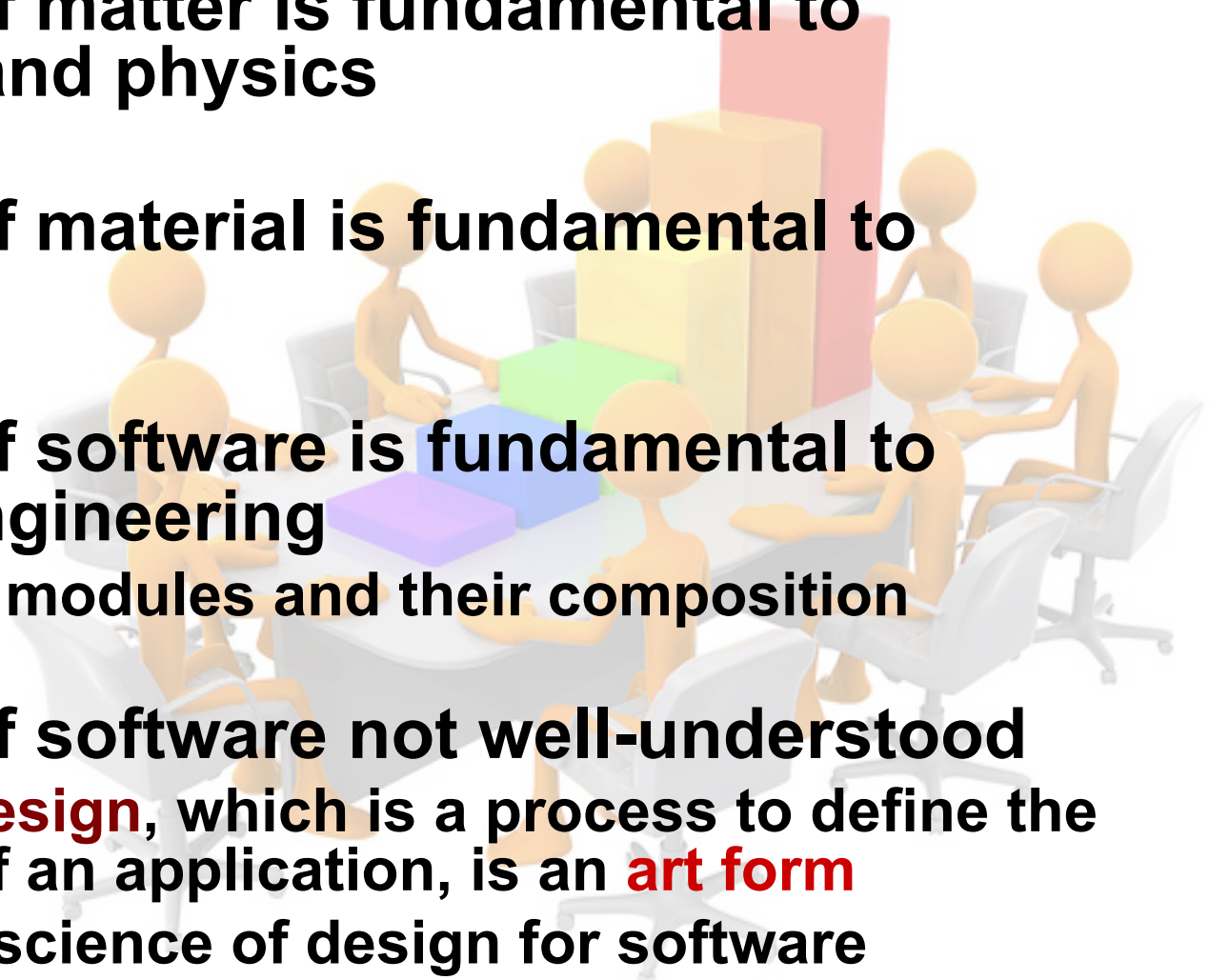
- **Generative Programming (GP)**
 - Understand domain well enough to generate software
 - Programs that synthesize other programs
 - Metaprogramming, skeleton communities

- **Domain-Specific Languages (DSLs)**
 - Elevate program specifications to compact domain-specific notations that are easier to write & maintain

- **Automatic Programming**
 - Culmination of GP and DSLs
 - Generate efficient programs from declarative specs
 - Precursor to Model-Based Software Engineering



A Case for Automated Programming

- Structure of matter is fundamental to chemistry and physics
 - Structure of material is fundamental to engineers
 - Structure of software is fundamental to software engineering
 - structure = modules and their composition
 - Structure of software not well-understood
 - **Software design**, which is a process to define the structure of an application, is an **art form**
 - We need a science of design for software
- 

So Did the Database People Get it?

CODD, E. F.

A relational model of data for large shared data banks. *Comm. ACM* 13, 6 (June 1970), 377-387.

In this article, the author discusses the advantages of using a relational rather than a network model for representing data in a data management system. It is his contention that such a model would provide greater independence of the data from application programs and would provide a means of making consistency checks on facts in the database.

The author describes a convention under which most data structures can be represented as sets of relations. He then describes how some of the common set operations correspond to database operations.

While the relational model is not a new one, this paper probably contains its most rigorous and elegant statement and, as such, represents a valuable contribution to the theory of data processing.

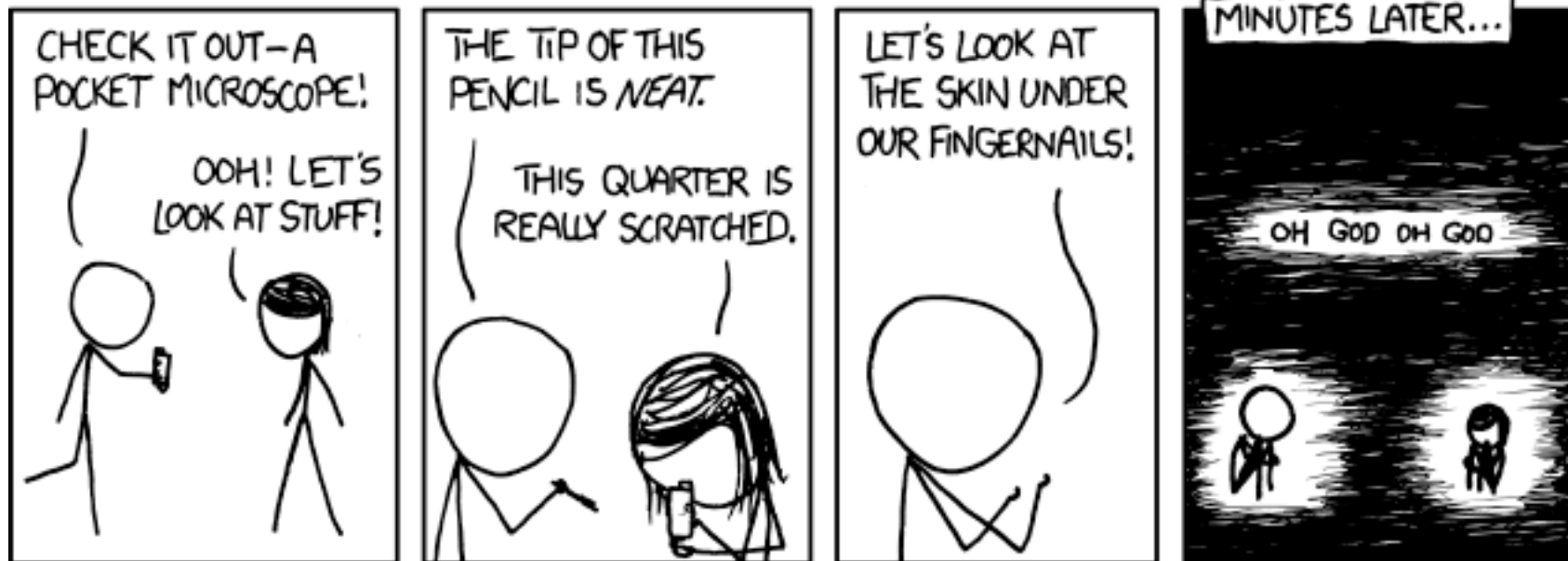
However, it would appear to this reviewer that the model is of more theoretical than practical interest.

While the relational model does indeed eliminate some data dependencies (e.g., ordering dependence) it does so by eliminating the practical advantages of ordering. While it does provide a mechanism for automatically making some data checks, these checks would often be so computationally painful that they would be of little practical value.

R. W. Elliott, College Station, Texas
#20,780 March 1971

- Nope...Look at the Computing Surveys Review of Codd's 1970 seminal paper on the Relational Model...
- Do you get it?
- This is a race and my leg is running now...
 - Your leg is coming up!

Looking at the details





Key Original Papers

■ Automatic Programming

Robert Balzer. **A 15-year Perspective on Automatic Programming.** *IEEE Transactions on Software Engineering*, 11(11):1257–1268, November 1985.

David R. Barstow. **Domain-Specific Automatic Programming.** *IEEE Transactions on Software Engineering*, 11(11):1321–1336, November 1985.

Charles Rich and Richard C. Waters. **Automatic Programming: Myths and Prospects.** *IEEE Computer*, 21(8):40–51, August 1988.

■ Automated Planning

Keith Golden. **A Domain Description Language for Data Processing.** Proc. of the International Conf. on Automated Planning and Scheduling, 2003.

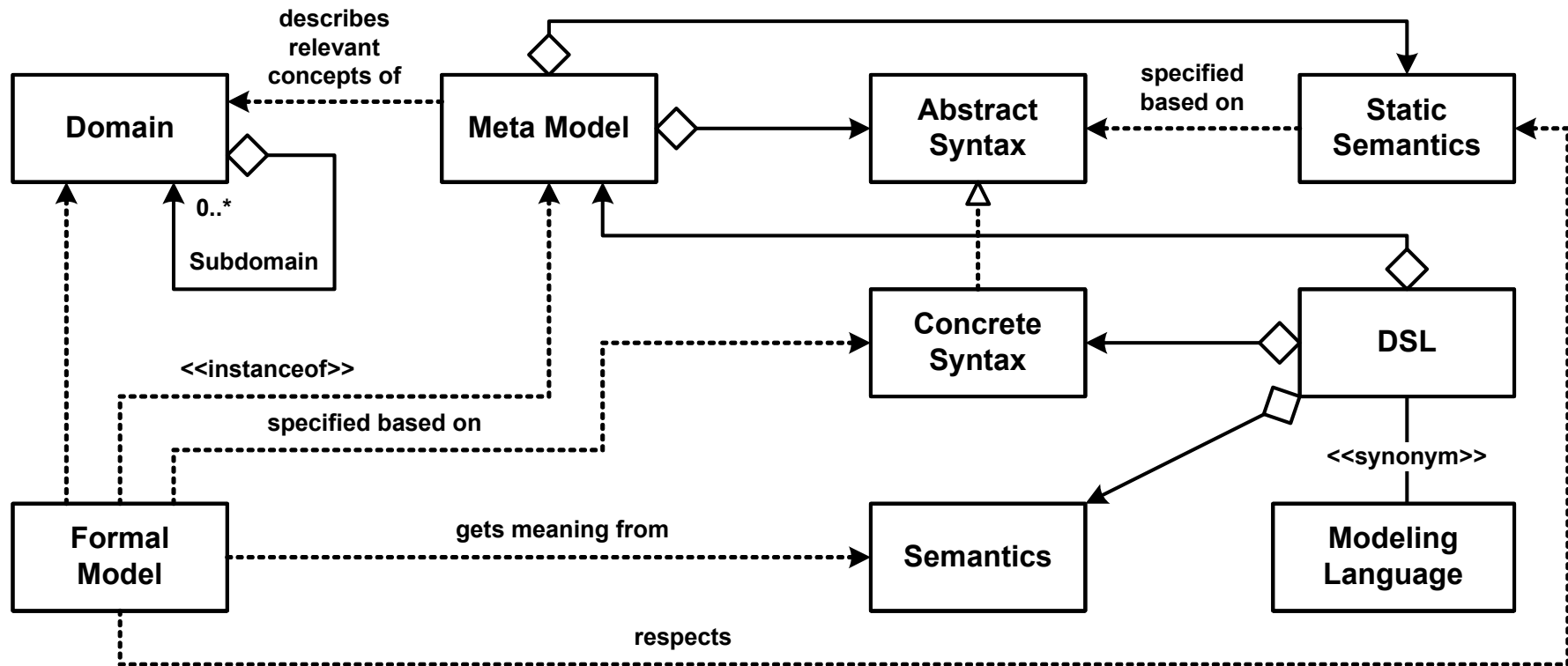
M. Stickel et al. **Deductive Composition of Astronomical Software from Subroutine Libraries.** Proc. of the International Conf. on Automated Deduction, 1994.

If you automated software production, what would the factory contain?

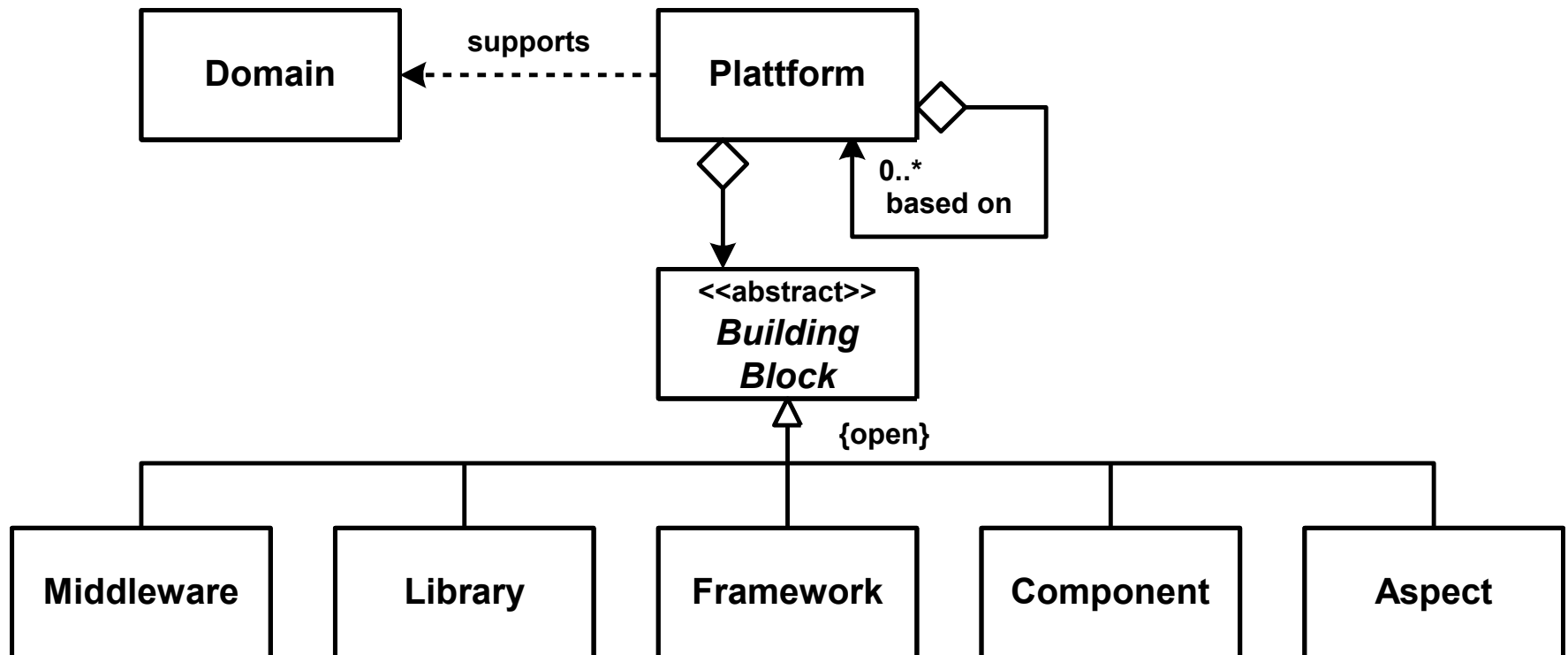
- Think for 15 seconds...
- Let's talk...



Recall: Modeling Concepts

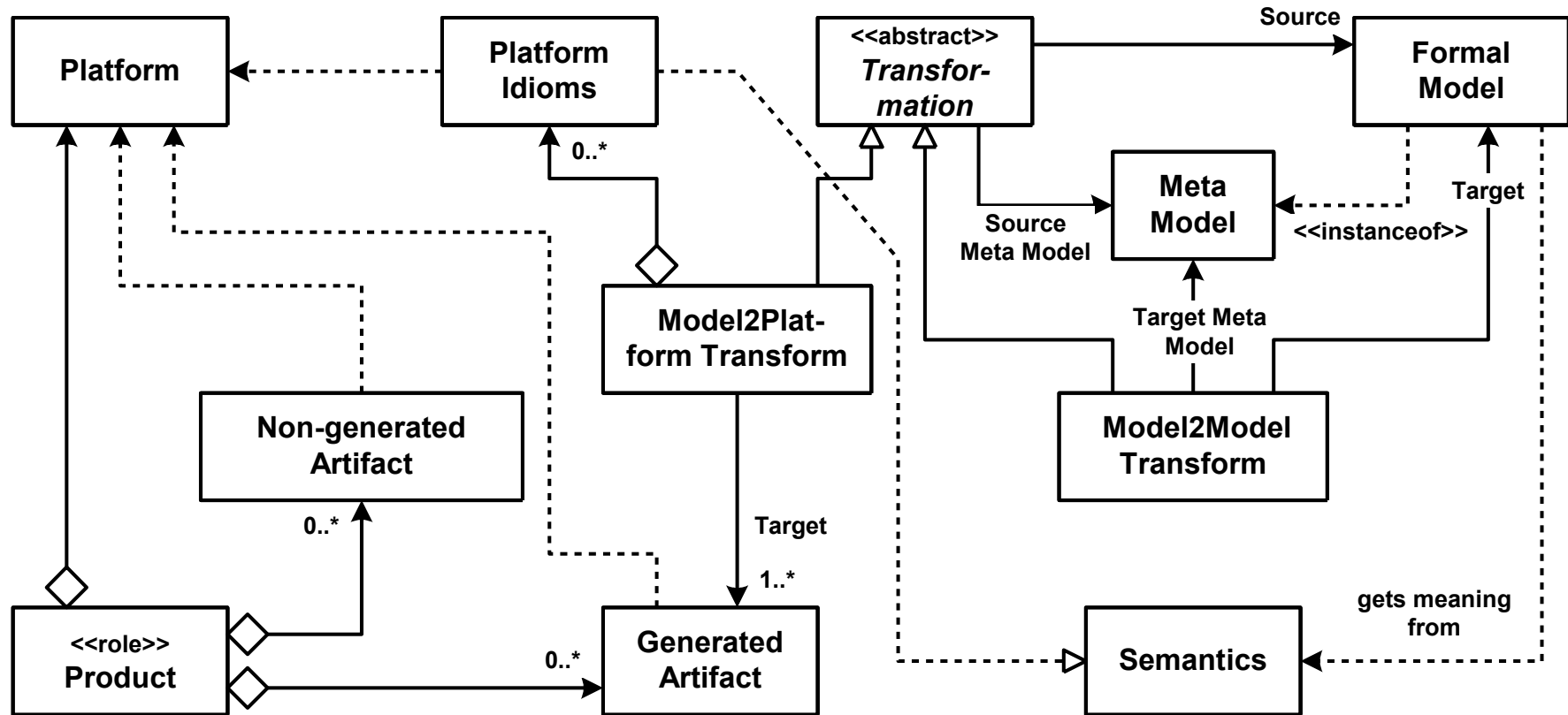


Recall: Platform Concepts

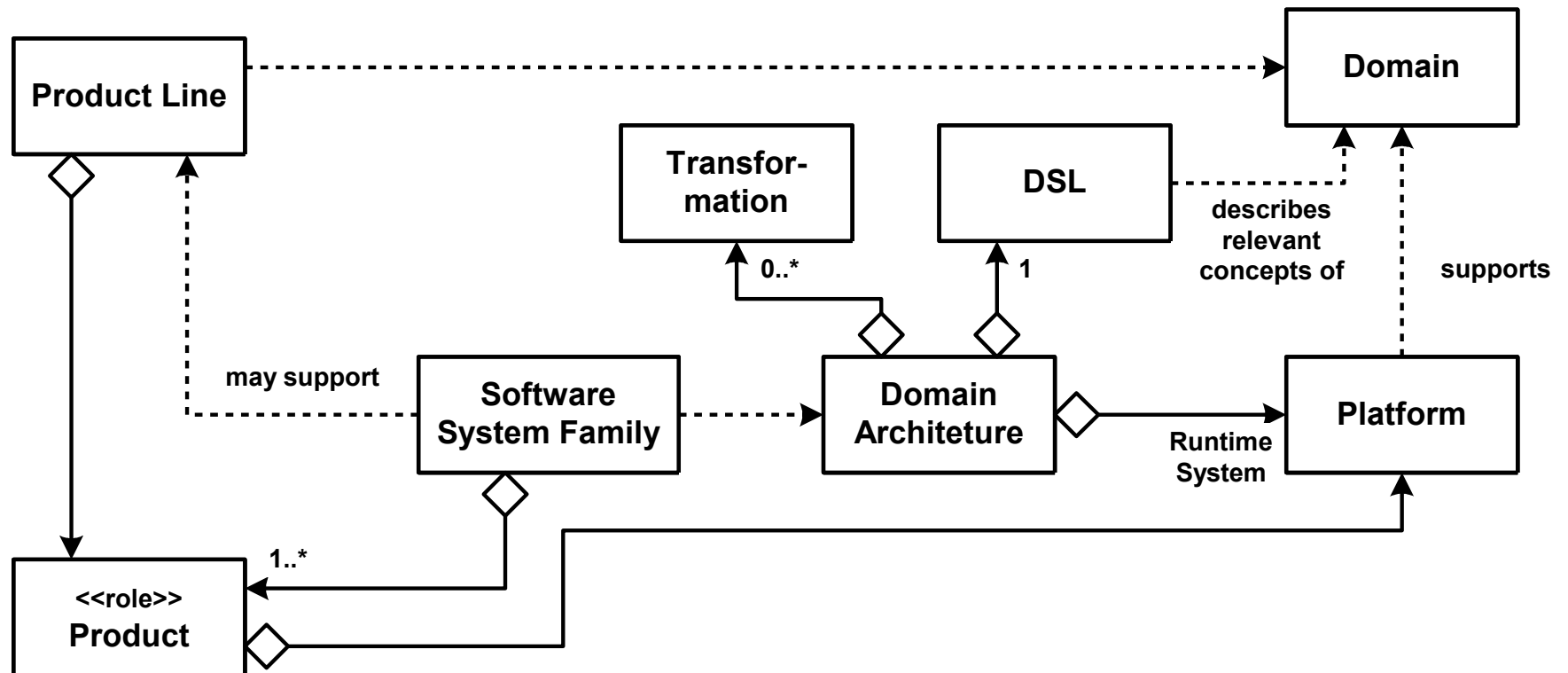




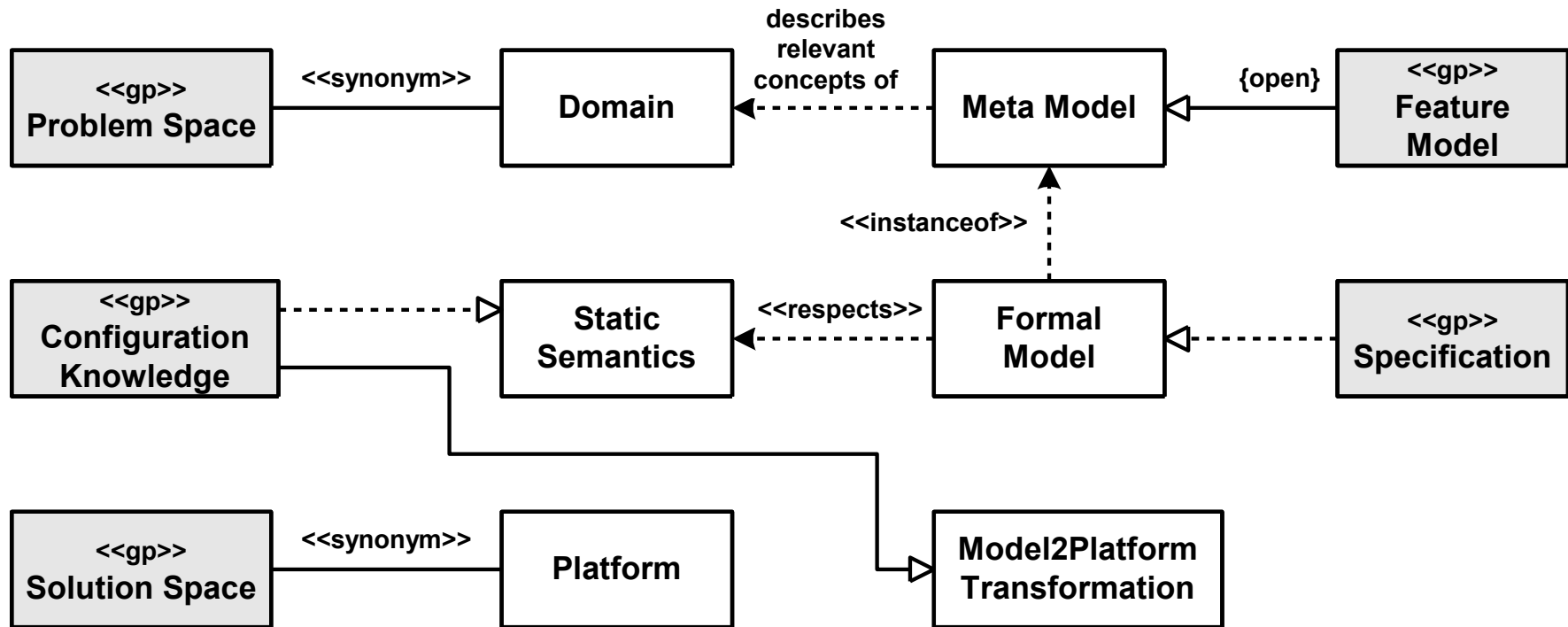
Recall Transformations



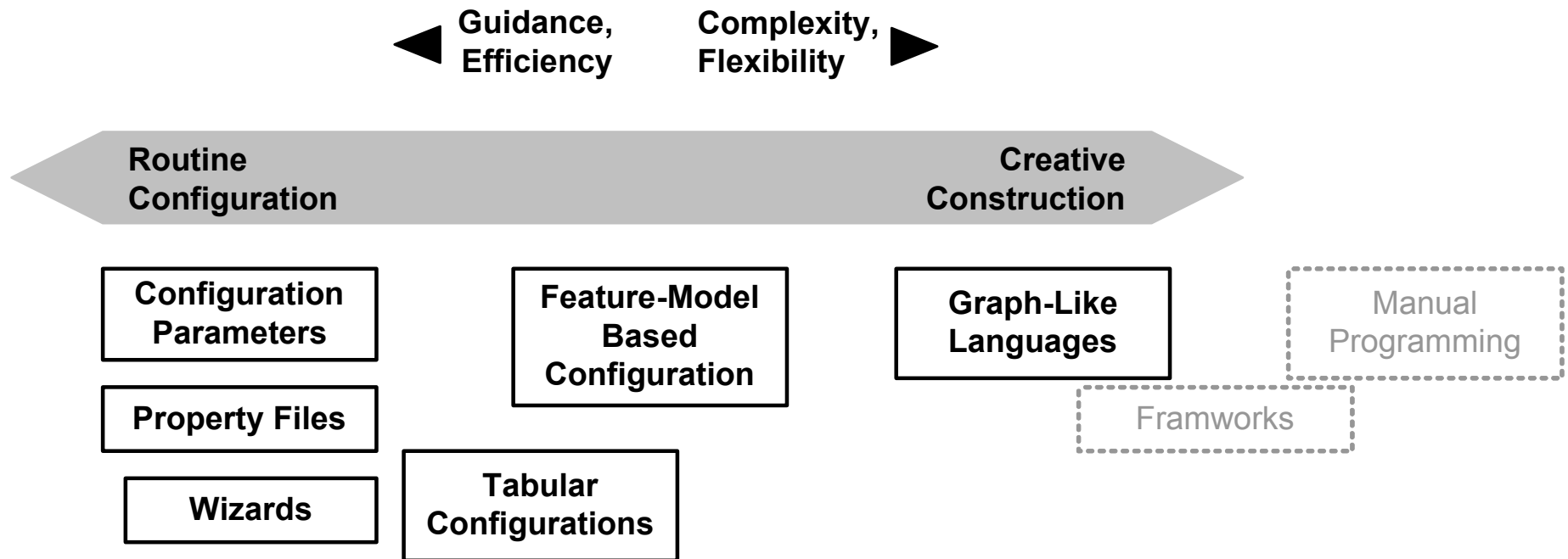
Recall: Software System Product Lines



Generative Programming Concepts



More GP Concepts



- Platform consists of maximally combinable and minimally redundant components
- GP is often routine configuration as opposed to creative construction

Software Factories

- 1990's Software Factories emerged as the new automated programming
- Faced an untrained community coupled with limitations in computing capabilities
 - The Virtual Software Factory
 - Software Templates
 - Software Refinery
- Devolved into IDEs configured for efficient development of Domain applications (led by Microsoft these days)

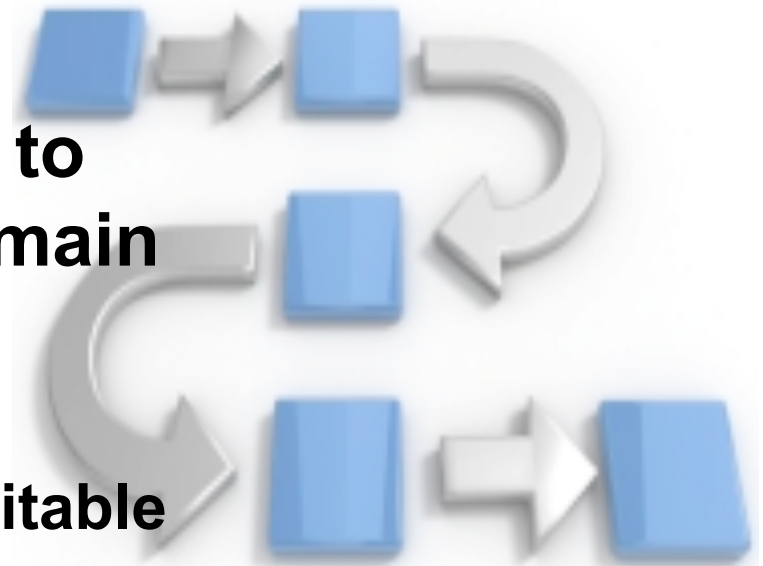


Software Factories Schema

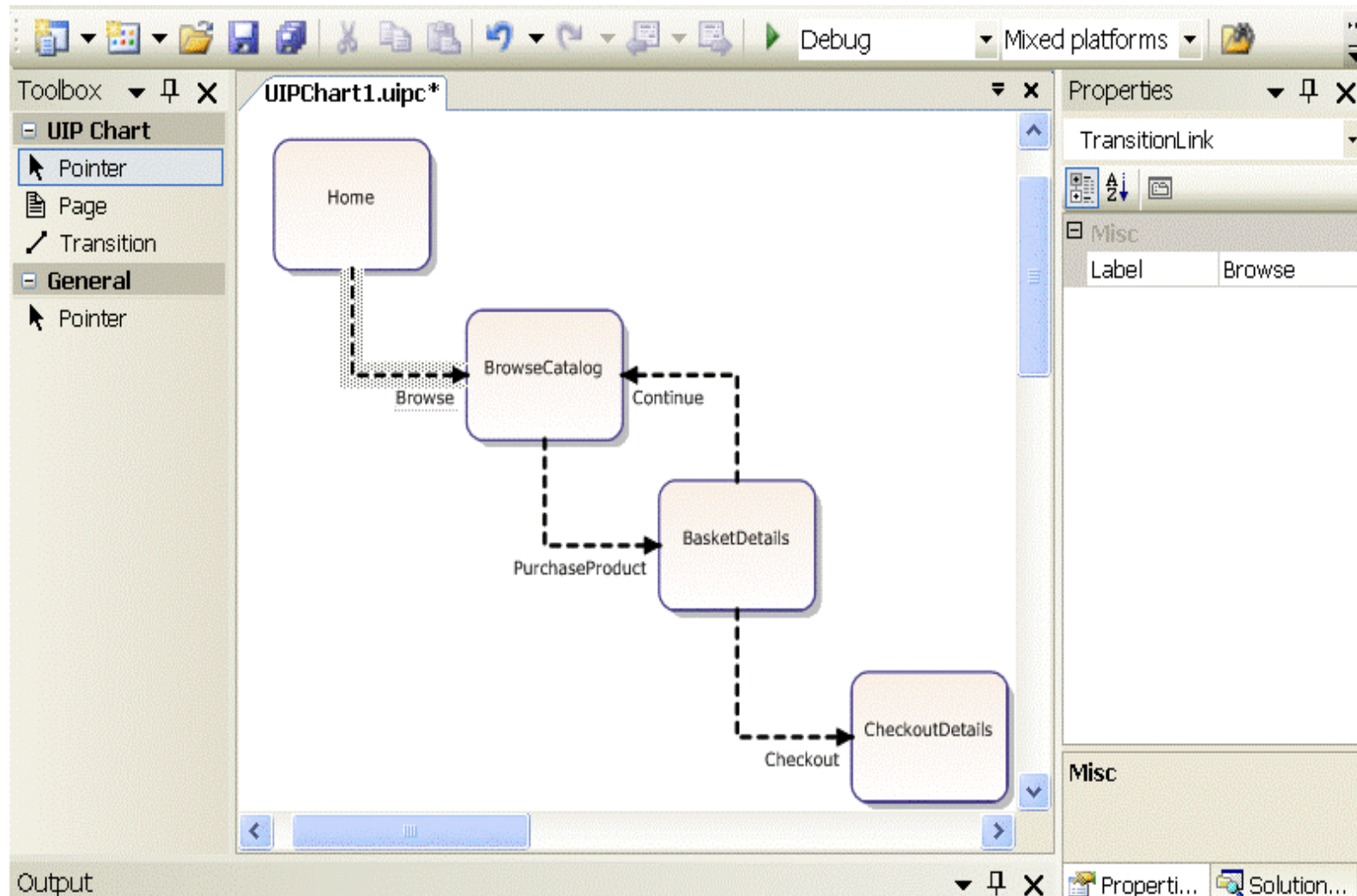
- Schema defines viewpoints for modeling and building a system (e.g., enterprise system):
 - Presentation, form layout and workflow
 - Component structure and business data model
 - Persistence mapping, Deployment, ...
- Schema identifies core artifacts as well as the most efficient way of producing them
 - DSLs, frameworks, patterns, manual programming
- Schema identifies commonalities and differences among applications in the domain

Software Factories Templates

- **Makes the Schema usable**
- **Load SF Template into IDE to configure it for specific domain**
 - Provides the necessary frameworks or libraries
 - Contributes project types suitable for the factory
 - Delivers build scripts
 - Extends IDE with DSL editors and transformations



MS DSL Tools Example



Defining a Metamodel

The screenshot displays the Visual Studio IDE with the ReseaudPetri.dmd metamodel design. The main window shows a class hierarchy and relationships. The class hierarchy includes ElementHasElement, PlaceHasJeton, PlaceHasTransition, and TransitionHasPlace, all inheriting from RDP. The Element class has a ReferencedE relationship to another Element. The Jeton class has a Jeton relationship to another Jeton. The Place class has a Transition relationship to another Place and a TPlace relationship to another Place. The Transition class has a Transition relationship to another Transition. The Solution Explorer on the right shows the project structure, including Resources, Shell, and Templates. The Output window at the bottom shows the build logs, indicating a successful build.

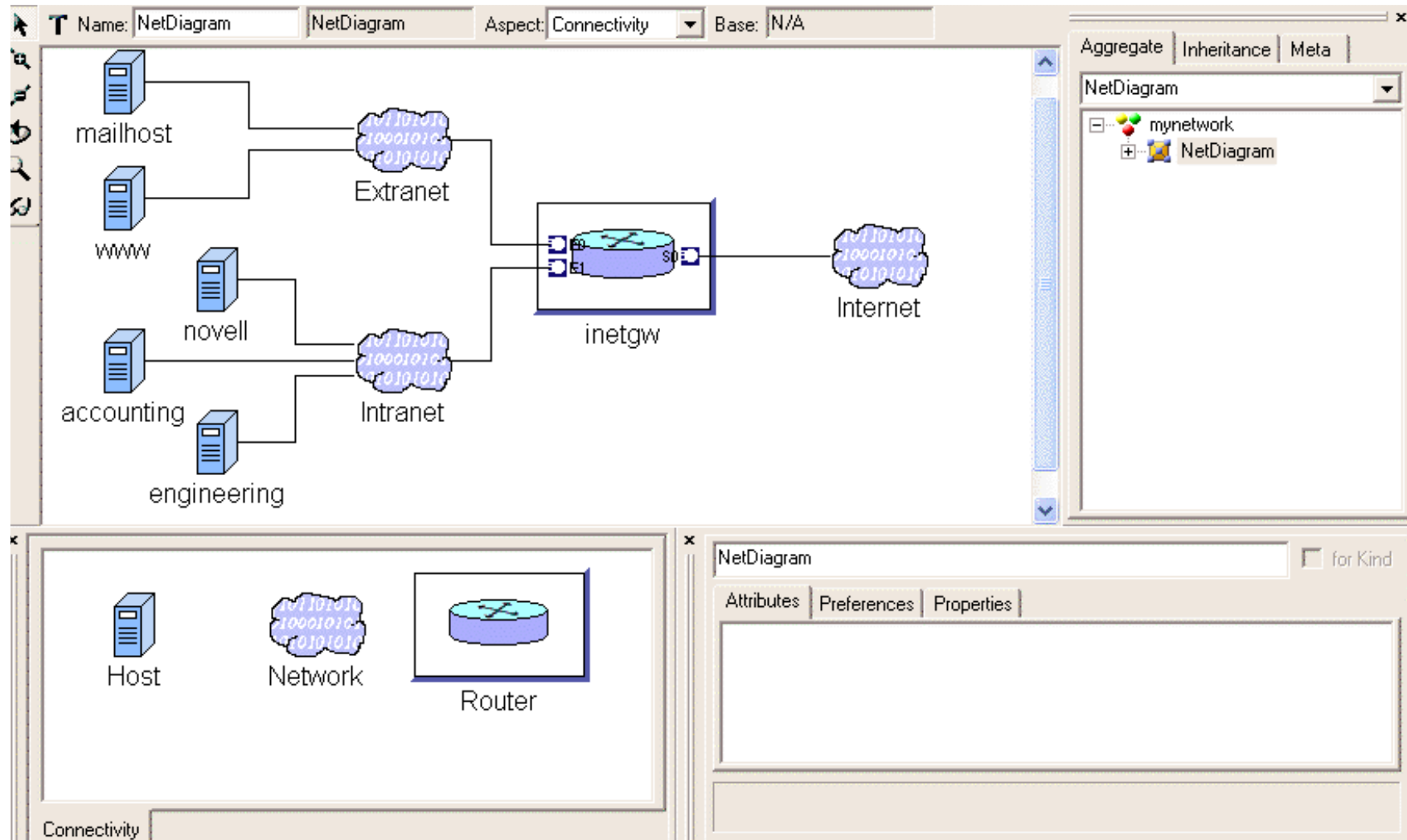
```
ToolWindow: Example.ReseaudPetri.Designer.ReseaudPetriModelExplorer, {c5bd4cf5-1704-4ca7-9b70-8b946a763781}
Editor Extension: .rdp, {a29fab9e-fa92-4582-8450-8cd792d85bab}
LoadKey: ReseaudPetriPackageName
Version 1.0.0.0
Edition Required: Standard
SUCCEEDED: Example.ReseaudPetri.Designer
===== Build: 1 succeeded, 0 failed, 2 up-to-date, 0 skipped =====
```

Graphical Model Editor

The screenshot displays a graphical model editor interface with several key components labeled:

- Menubar:** Located at the top left, containing menu items like File, Edit, View, Window, and Help.
- Toolbar:** A row of icons below the menubar for various editing functions.
- Modebar:** A vertical bar of icons on the left side of the main workspace.
- Model Editing Windows:** Two overlapping windows showing model diagrams. The top window, titled "System", shows a flow from "PreProcessing" to "Processing" to "PostProcessing". The bottom window, titled "Processing", shows a detailed view of the "Processing" block with inputs "In" and "Time", and outputs "Branch0" and "Branch1".
- Model Browser:** A tree view on the right side showing the hierarchical structure of the model, including folders like "System", "PostProcessing", "PreProcessing", and "Processing", along with specific components like "Branch0", "Branch1", "Freq", "In", and "Time".
- Partbrowser:** A panel at the bottom left showing available components: "CompoundParts", "InputSignals", "OutputSignals", and "PrimitiveParts".
- Attribute Browser:** A panel at the bottom right showing properties for the selected "Branch0" component, including "Firing" (IFALL), "Script" (ComputeTime), and "Priority" (10).
- Statusbar:** Located at the very bottom, showing the current state as "Ready" and system information like "EDIT 100% SF2000 11:39 AM".

GME: Modeling based on previously defined MM



GME: OCL Constraint Validation

T Name: SF2000 | ParadigmSheet | Aspect: Constraints | Base: N/A

```

classDiagram
    class Folder["Folder <<Folder>>"]
    class Processing["Processing <<Model>>"]
    class Primitive["Primitive <<Model>>"]
    class Compound["Compound <<Model>>"]
    class Signal["Signal <<Atom>>"]
    class InputSignal["InputSignal <<Atom>>"]
    class OutputSignal["OutputSignal <<Atom>>"]
    class DataflowConn["DataflowConn <<Connection>>"]
    class ParameterConn["ParameterConn <<Connection>>"]

    Folder <|-- Processing
    Processing <|-- Primitive
    Processing <|-- Compound
    Signal <|-- InputSignal
    Signal <|-- OutputSignal
  
```

AtLeastOnePart for Kind

Attributes	Preferences	Properties
Description:	Compounds must have parts	
Default parameters:		
Equation:	self.parts()->size > 0	
Priority (1=High):	2	
Depth:	1	
On close model	False	
On create	False	
On delete	False	
On new child	False	
On lost child	False	
On move	False	
On derive	False	
On connect	False	
On disconnect	False	
On change attribute	False	
On change property	False	
On change assoc.	False	
On refer	False	
On unrefer	False	
On include in set	False	
On exclude from set	False	



Homework and Milestone Reminders

- **Milestone 3: Light-Weight Transformation Environment (see Milestone 3 assignment)**
 - Due by 11:55pm, Thursday, May 12th, 2011.

- **Milestone 4: Final MBSE Environment (see Milestone 4 assignment)**
 - Due by 11:55pm, Friday, May 13th, 2011.