MDA Distilled

Stephen J. Mellor
Vice-President
Project Technology, Inc.
http://www.projtech.com

Table of contents

1. What’s the problem?
2. Models
3. Metamodels
4. Mappings
5. Marks
6. Building a Language
7. Agile MDA
8. Conclusion

What’s the problem?

Software is expensive, and productivity is low for many reasons. Amongst them:

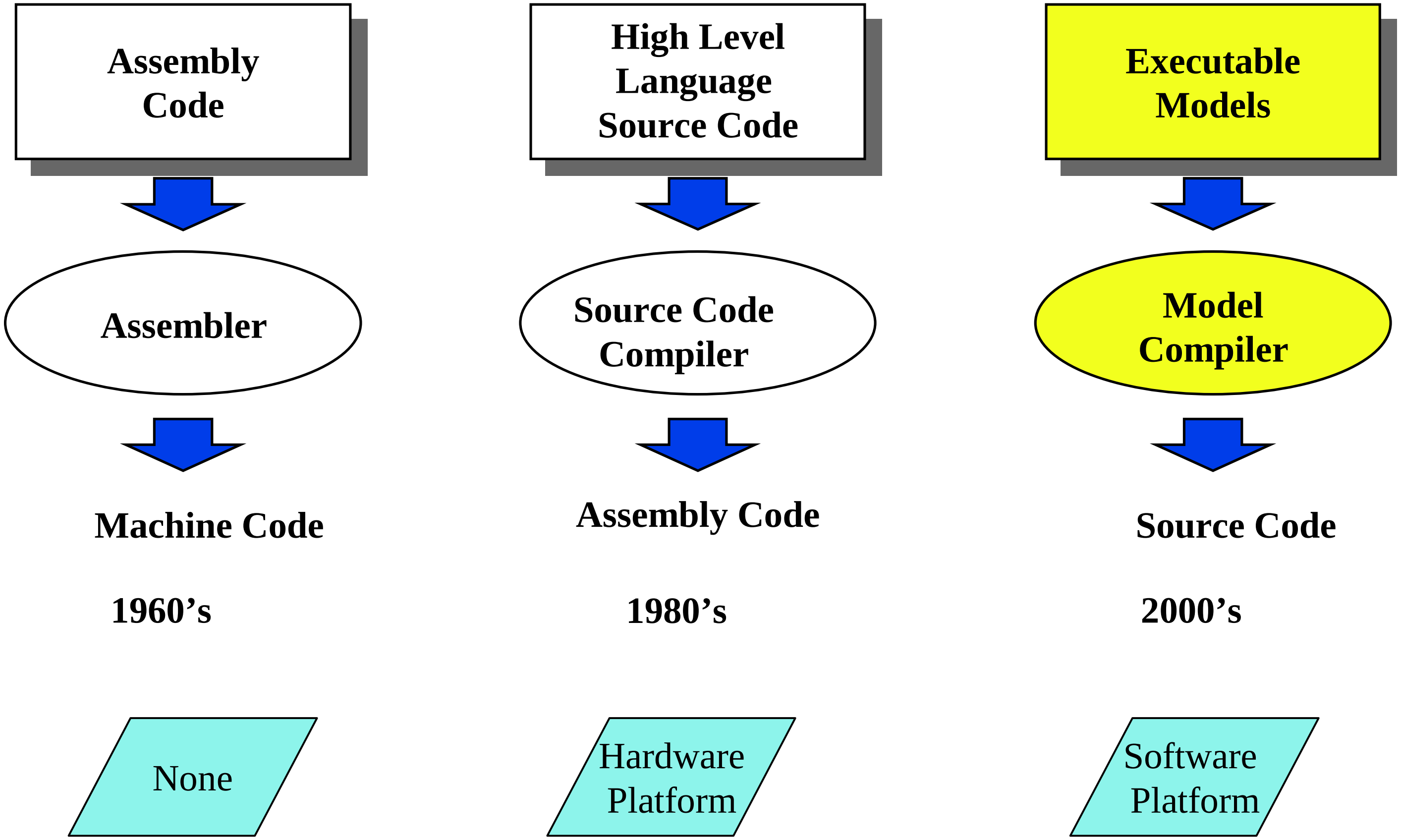
- Code is at too low level of abstraction
- Reuse occurs (to the extent it does at all) at too low a granularity
- Any code is glued together (at great expense) to its infrastructure (also expressed as code)
- Mapping information (design expertise) is applied—then lost

No wonder!

Expensive and hard-to-find!

Language abstraction

High-level language source code is two-dimensional.



Assembly Code

Assembler

Machine Code

1960’s

None

High Level Language Source Code

Source Code Compiler

Assembly Code

1980’s

Hardware Platform

Executable Models

Model Compiler

Source Code

2000’s

Software Platform

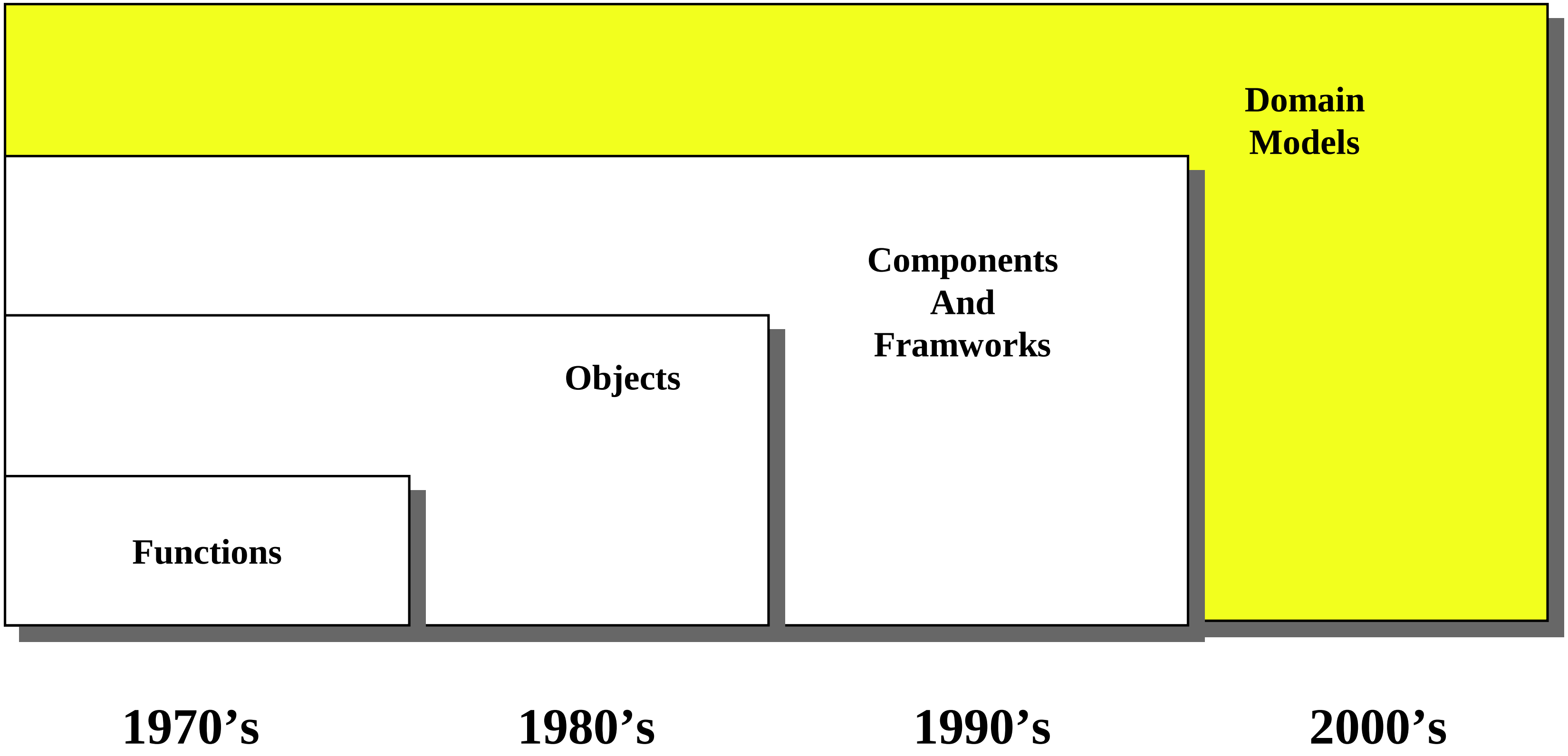
Sequential
1-D

Block-structured
2-D

Graphical
3-D

Reuse granularity

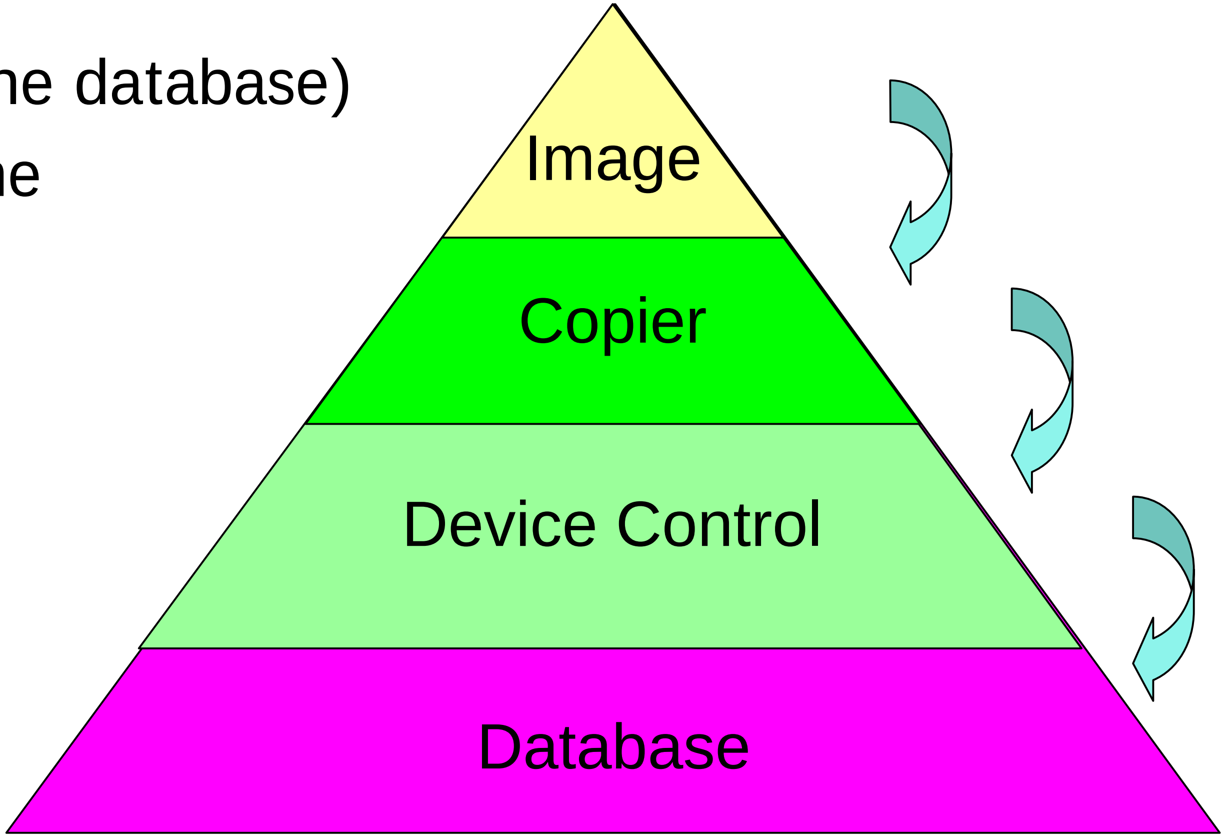
Components and frameworks require common infrastructure.



Code binds

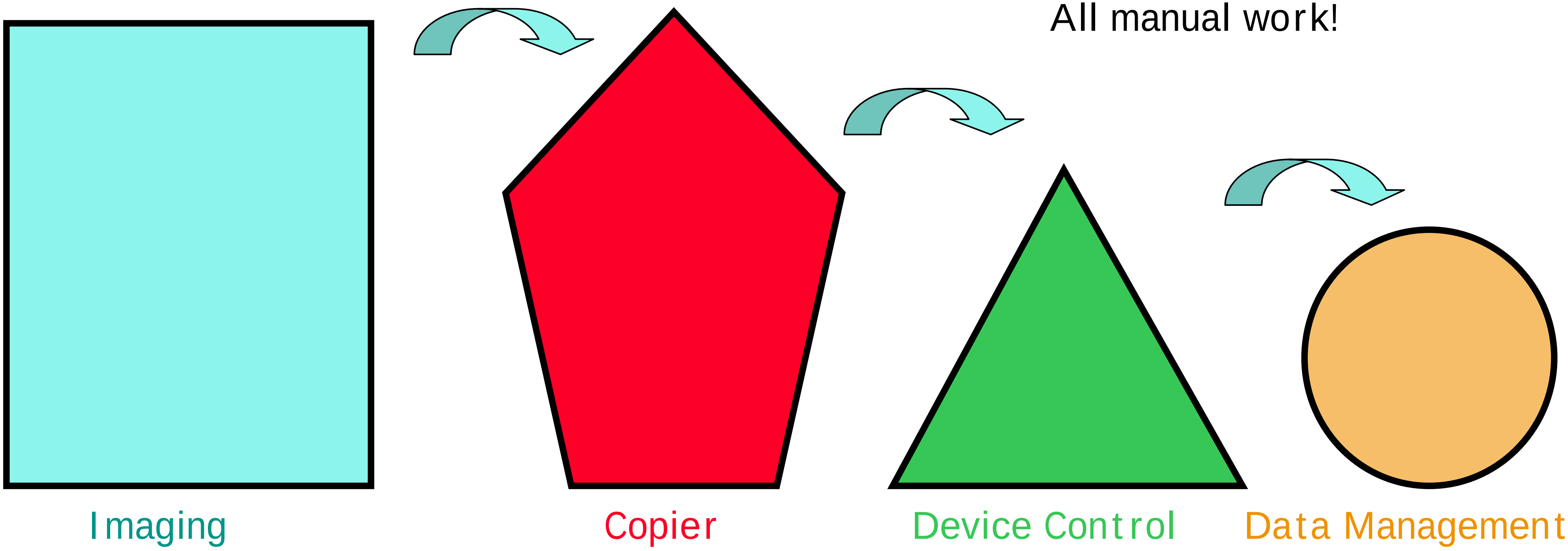
Code is glued to its infrastructure:

- Binds device control to the database
- Binds the copier to (device control and the database)
- Binds the image to the (copier and (device control and the database))...

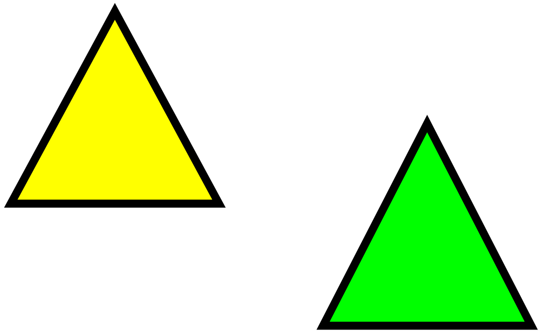


Mapping information is lost

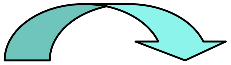
- Mapping between layers is all skilled manual labor.
- And once a mappings is ‘found,’ it is applied by hand
- When a change is made, the mappings are not repeatable.

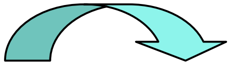


Components of an MDA solution



Capture *each layer* in a platform-independent manner as intellectual property.



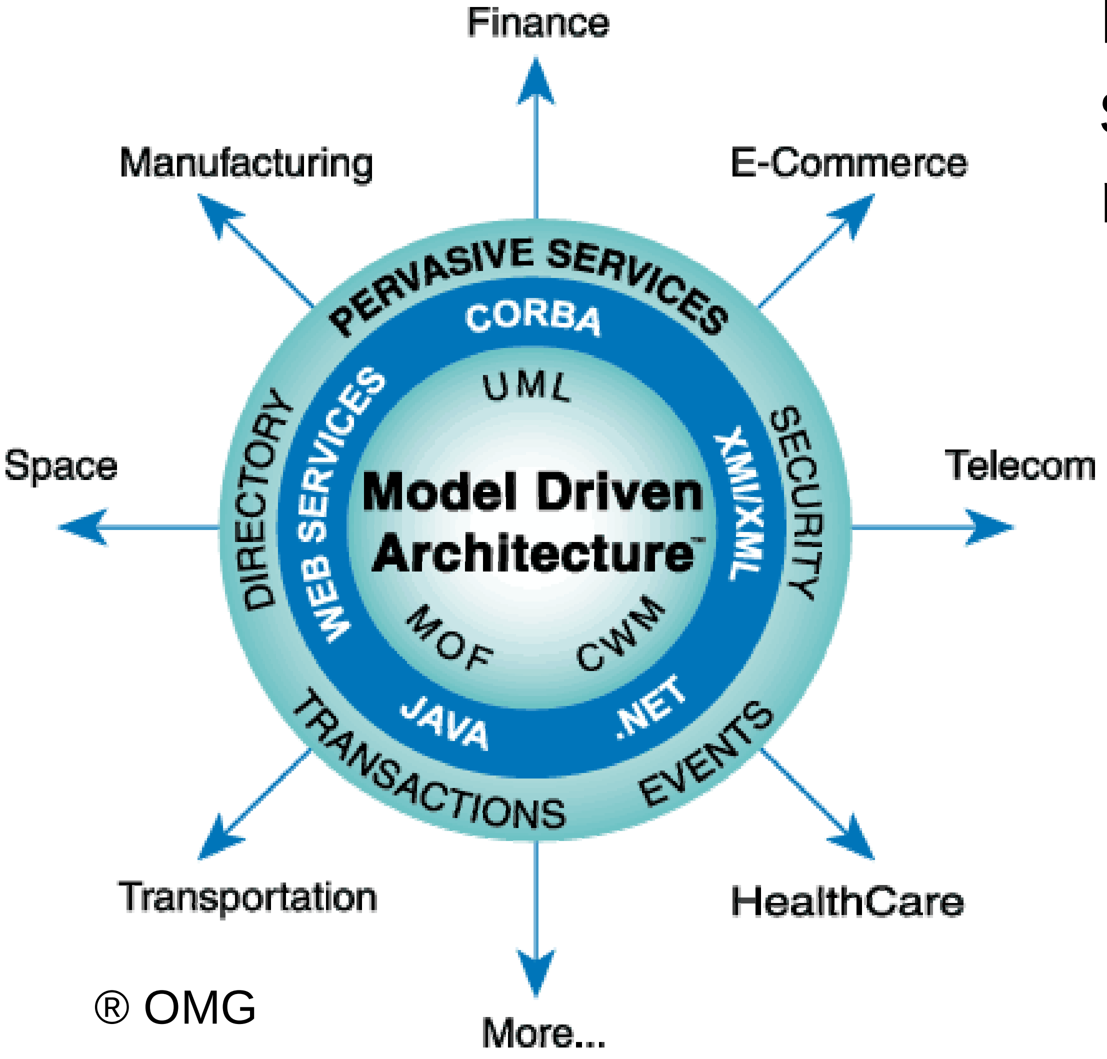


Capture *the mappings* to the implementation as intellectual property (IP).

Layer by layer.

Models and mappings become assets.

Enter Model-Driven Architecture

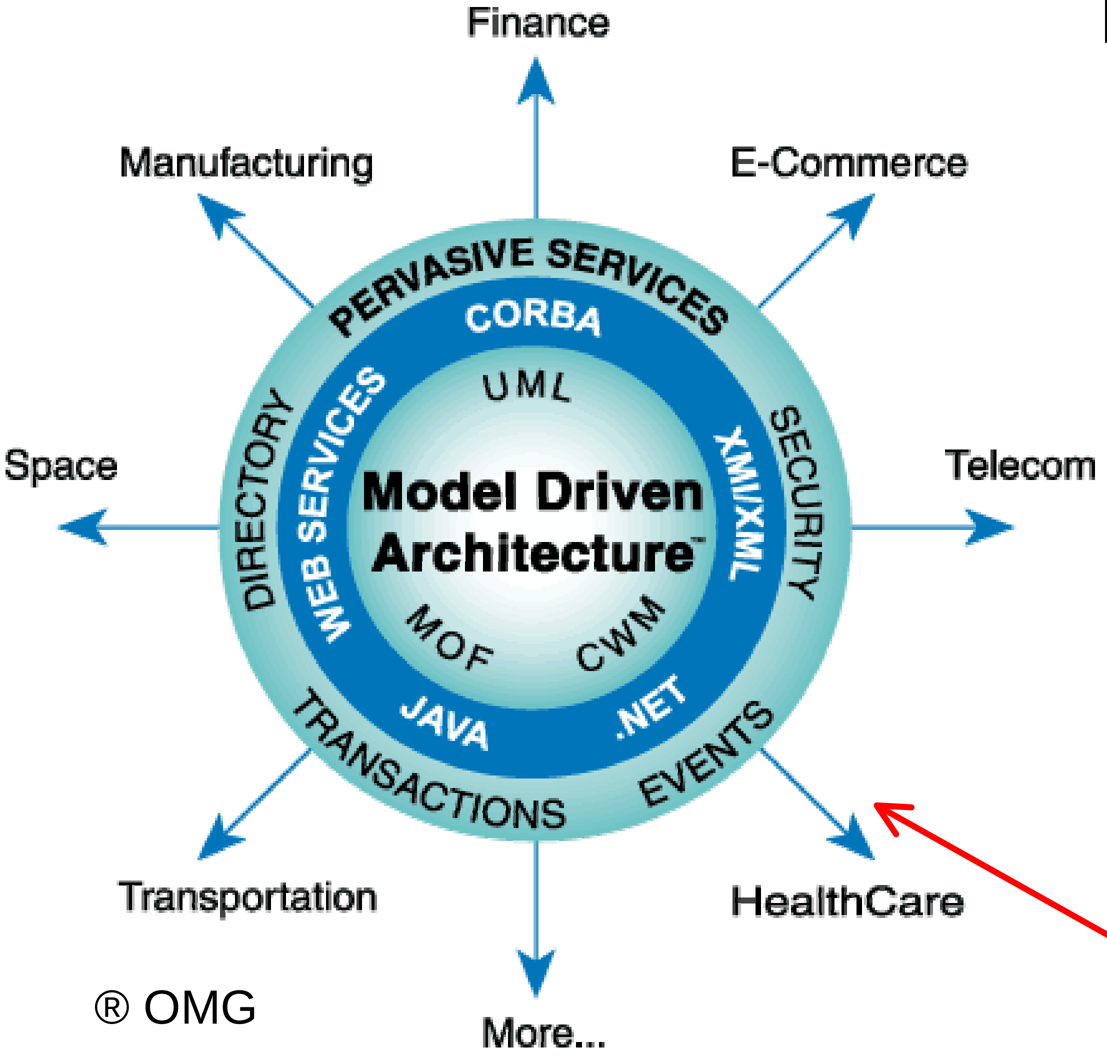


® OMG

MDA: an interoperability standard for combining models at design-time.

This enables a market for IP in software.

Enter Model-Driven Architecture



® OMG

MDA:

- Captures IP as models and enables protection of them
- Allows IP to be mapped automatically
- Allows multiple implementations
- Makes IP portable

This enables a market for IP in software.

Table of contents

1. What’s the problem?
2. Models
3. Metamodels
4. Mappings
5. Marks
6. Building a Language
7. Agile MDA
8. Conclusion

Modeling language for software

"The Unified Modeling Language is a language for specifying, constructing, visualizing, and documenting the artifacts of a software-intensive system."

The UML Summary

® Object Management Group

Why model?

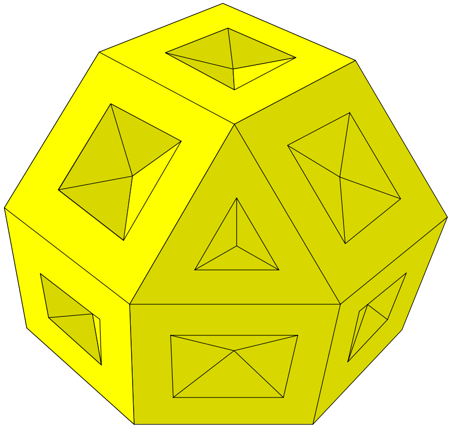
A good model:

- Abstracts away not-currently-relevant stuff
- Accurately reflects the relevant stuff, so it...
- Helps us reason about our problem
- Is cheaper to build than code
- Communicates with people
- Communicates with machines

What is a model?

A model is coherent set of elements that:

- Covers some subject matters
  - Doesn’t have to cover all subject matters
- At some level of abstraction
  - Doesn’t have to define realizations
- That need not expose everything
  - Doesn’t have to show everything at once
- That need not be complete in itself
  - Doesn’t have to include “code”



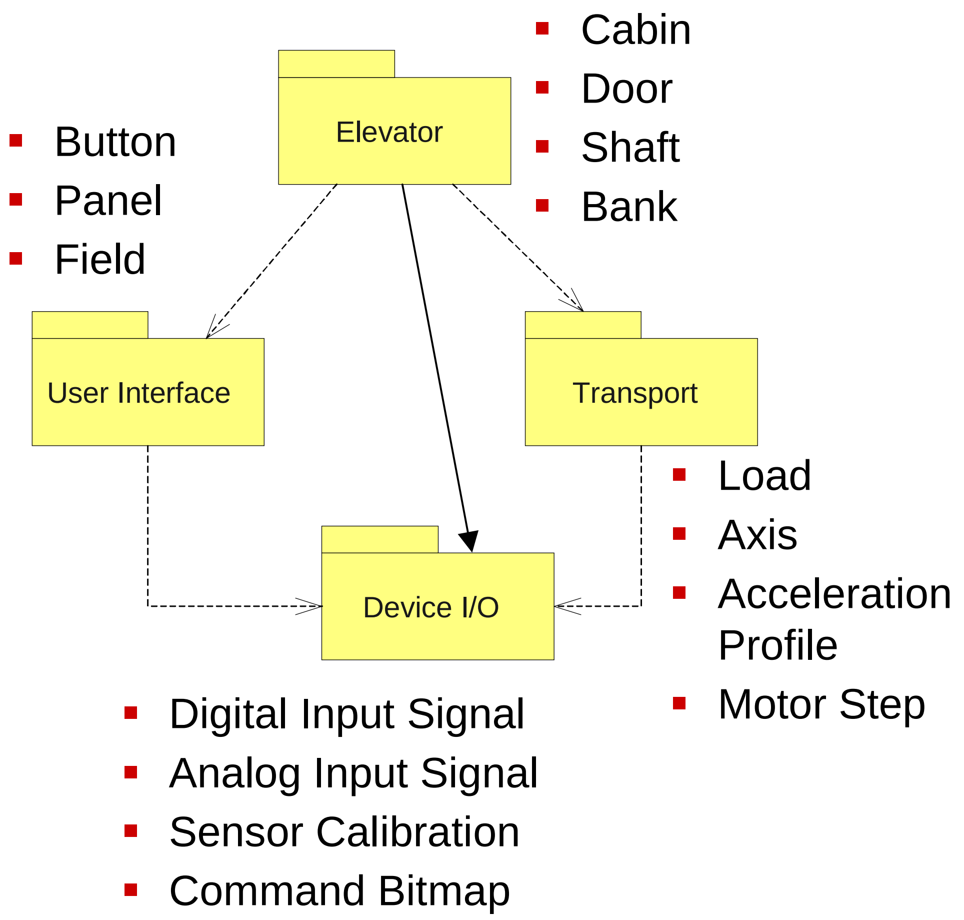
Seating plan?
Materials?
Interior?
No engine yet!

Subject matters

Good models come from separating layers by subject matter, so that each one is *platform independent.*

A change to models in one subject matter should not necessitate reconstruction of models in another subject matter.

A "PIM"



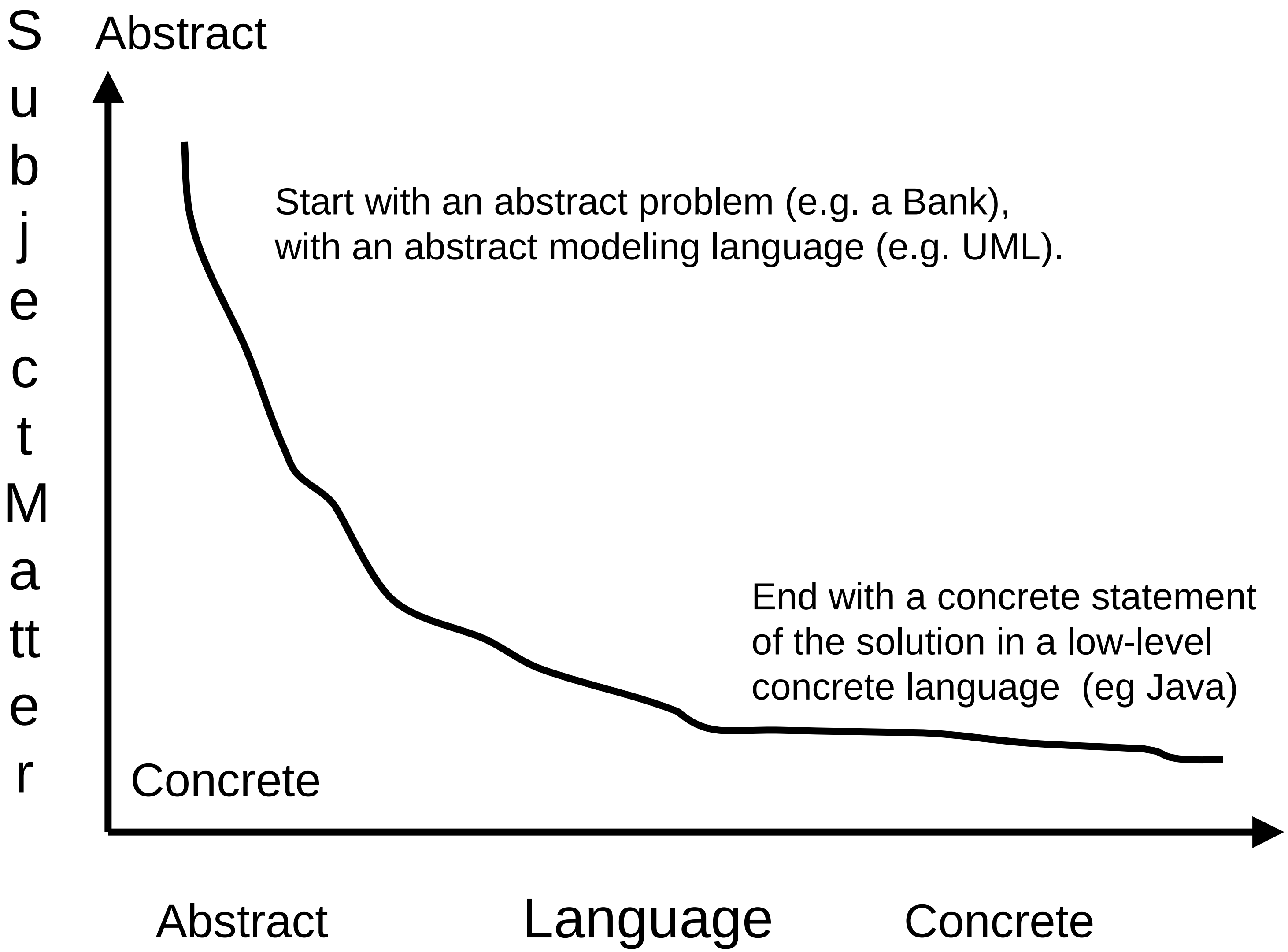
- Cabin
- Door
- Shaft
- Bank

- Button
- Panel
- Field

- Load
- Axis
- Acceleration Profile
- Motor Step

- Digital Input Signal
- Analog Input Signal
- Sensor Calibration
- Command Bitmap

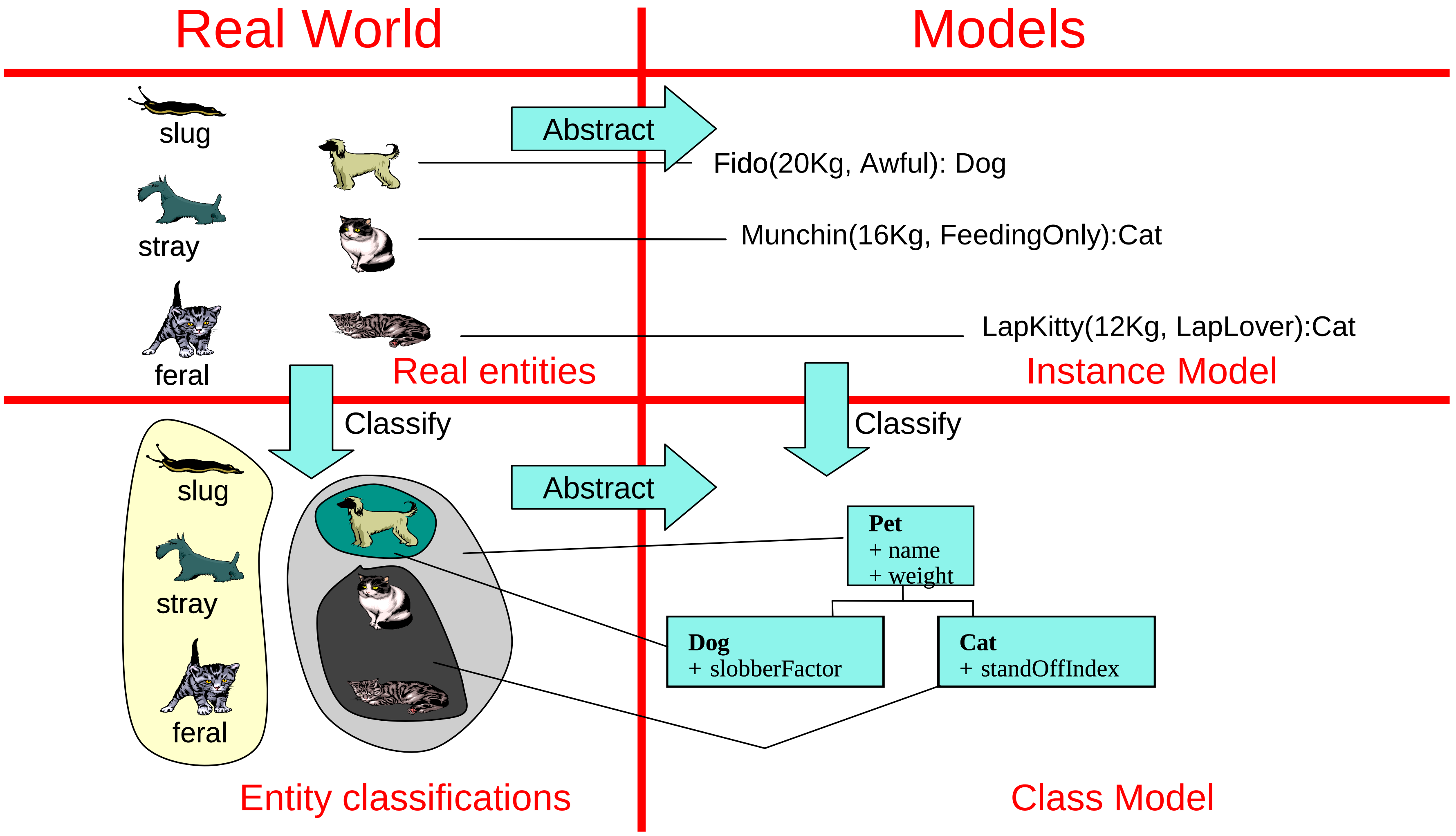
Language Abstraction



Start with an abstract problem (e.g. a Bank), with an abstract modeling language (e.g. UML).

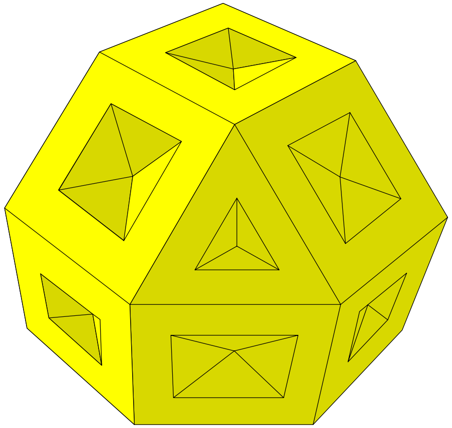
End with a concrete statement of the solution in a low-level concrete language (eg Java)

Abstraction and classification

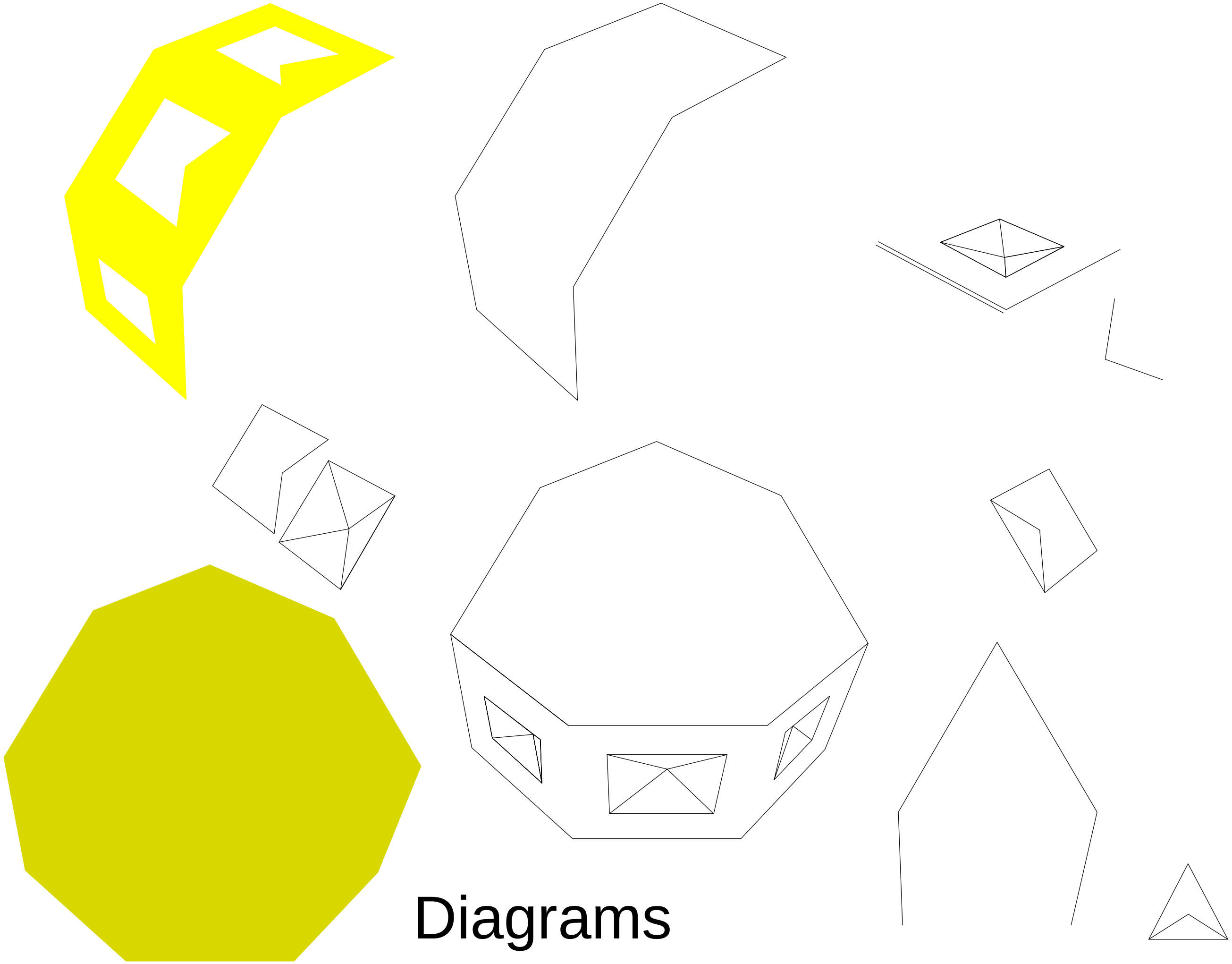


Model Views

A diagram is a coherent view on a model.



Model



Diagrams

Incompleteness

Code can be added to a model later.

Executable UML models

UML can be used as a semantic modeling language, if we:

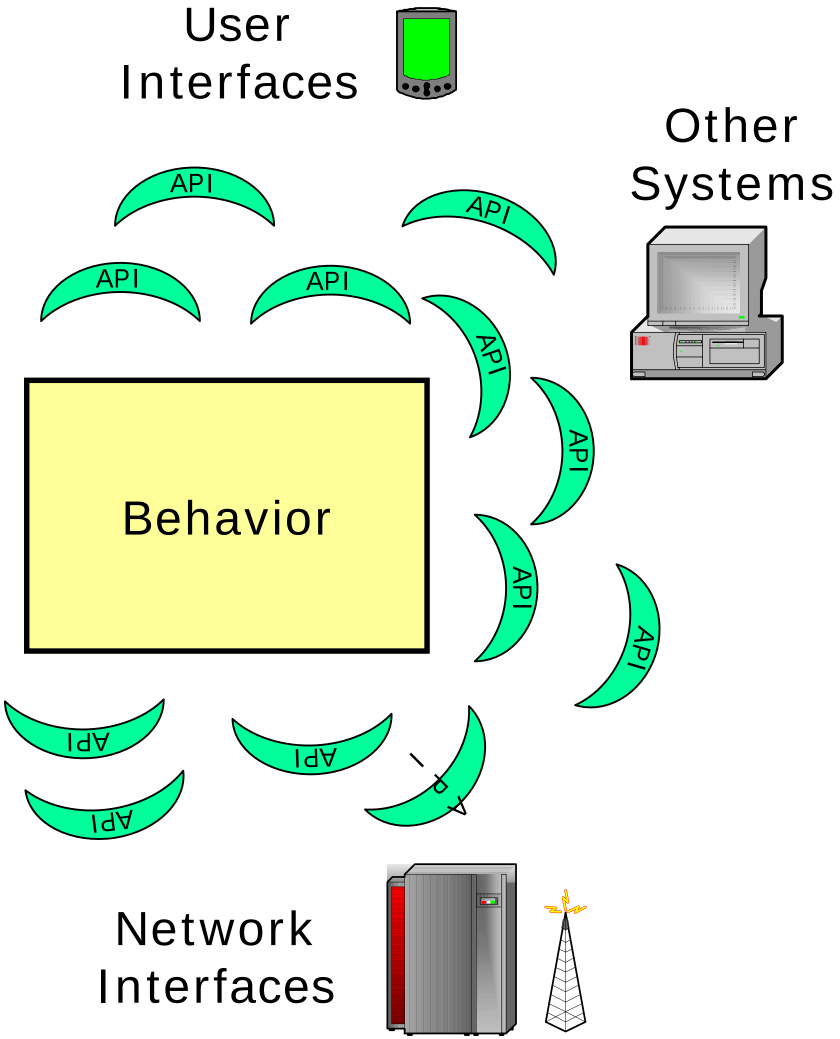
- Define actions
- Define the context
- Define execution rules

for an underlying semantic model.

The underlying semantic model is an:

executable
translatable
UML.

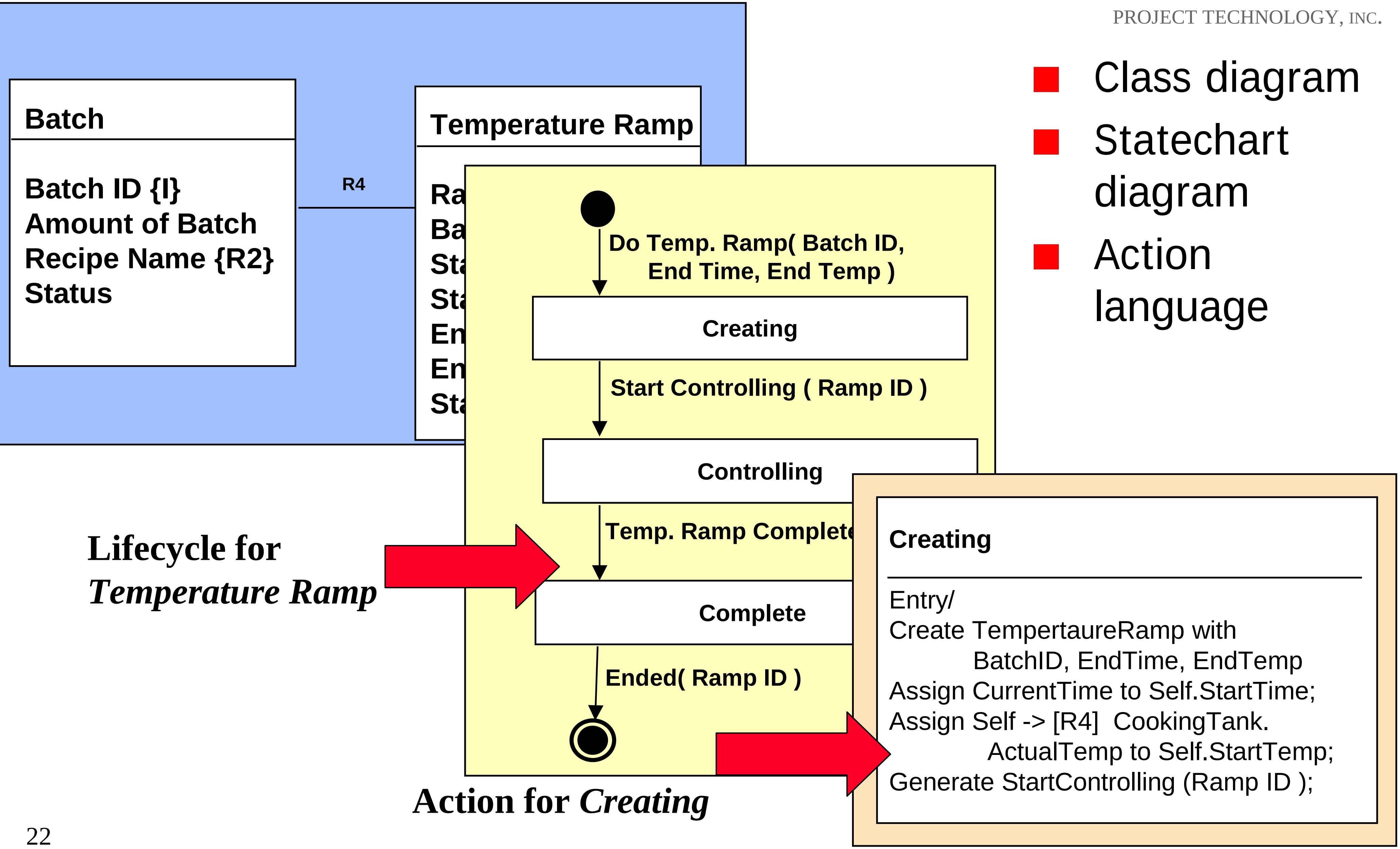
Defining behavior using UML



- UML can now be used to define behavior
  - UML 1.5/2.0 now has Action Semantics
- Use an executable translatable profile of UML ($^X{}_TUML$)
- $^X{}_T$UML defines behavior without making premature design decisions

Three primary diagrams

- Class diagram
- Statechart diagram
- Action language



```
Entry/
Create TempertaureRamp with
          BatchID, EndTime, EndTemp
Assign CurrentTime to Self.StartTime;
Assign Self -> [R4]  CookingTank.
            ActualTemp to Self.StartTemp;
Generate StartControlling (Ramp ID );
```

Lifecycle for
Temperature Ramp

Action for *Creating*

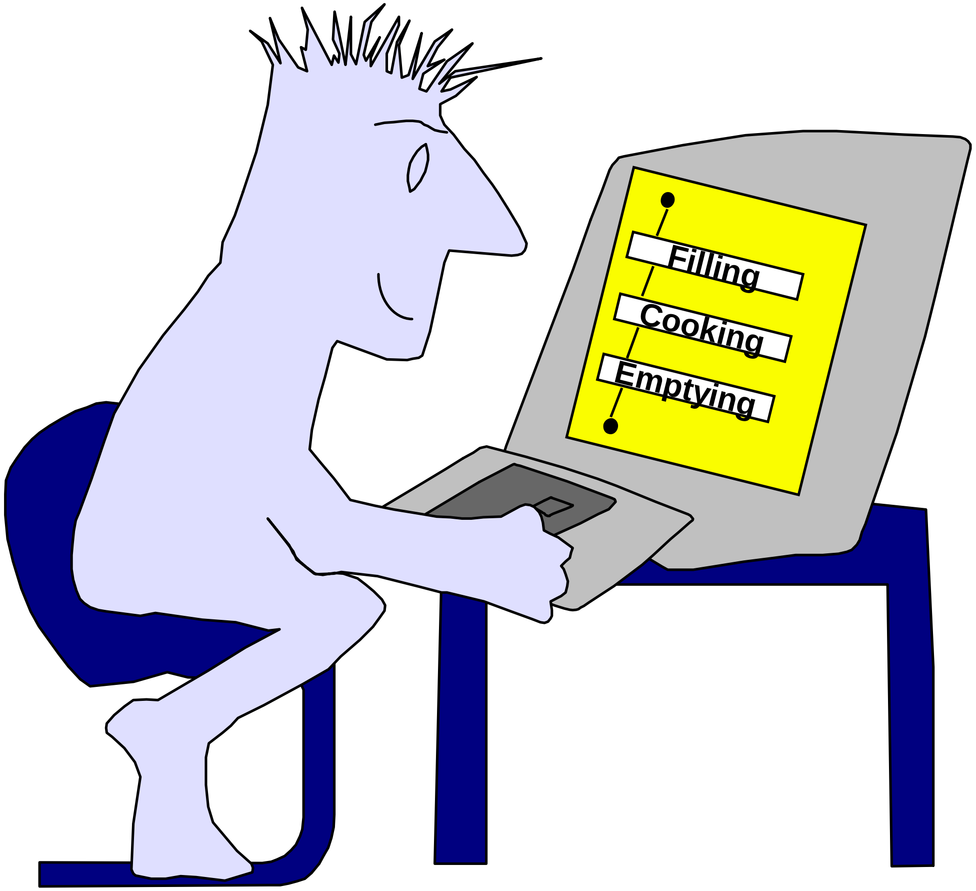
Table of contents

1. What’s the problem?
2. Models
3. Metamodels
4. Mappings
5. Marks
6. Building a Language
7. Agile MDA
8. Conclusion

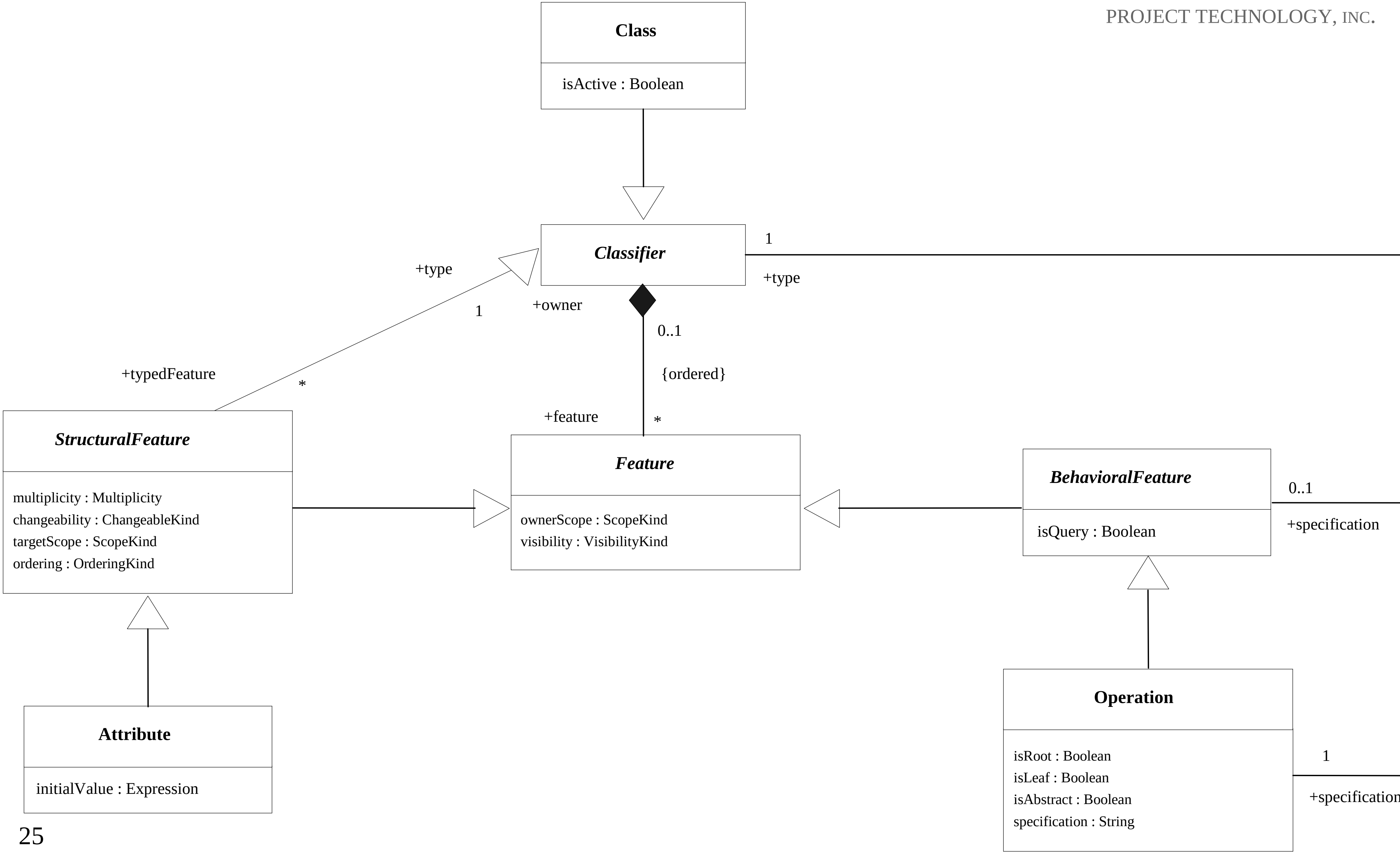
What is a metamodel?

A metamodel captures developer models in a model repository.

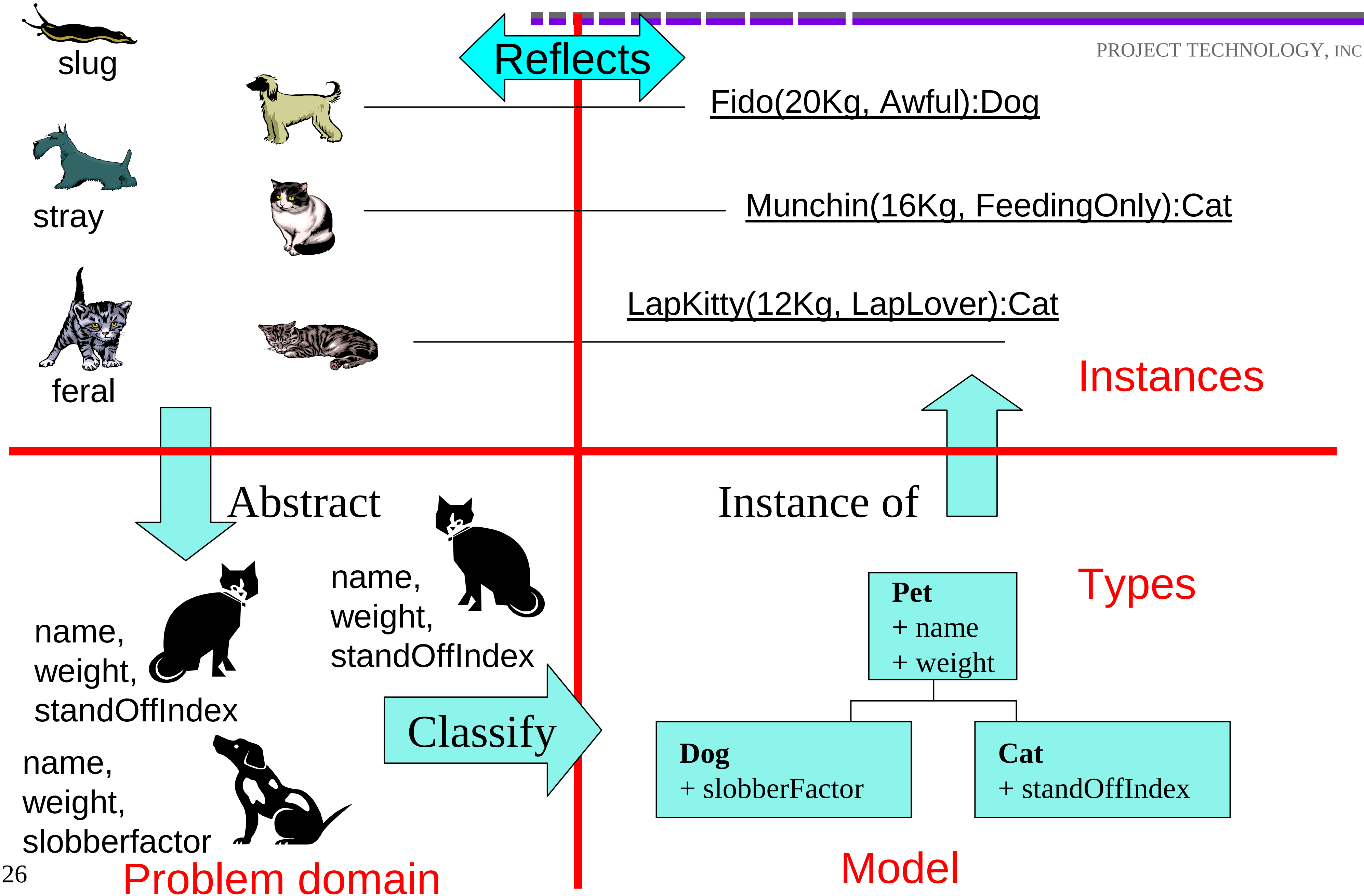
What is the structure of the repository?



UML metamodel



Instance-of FIX ME



Reflects

slug

stray

feral

Fido(20Kg, Awful):Dog

Munchin(16Kg, FeedingOnly):Cat

LapKitty(12Kg, LapLover):Cat

Instances

Abstract

Instance of

name,
weight,
standOffIndex

name,
weight,
standOffIndex

name,
weight,
slobberfactor

Classify

Types

Pet
+ name
+ weight

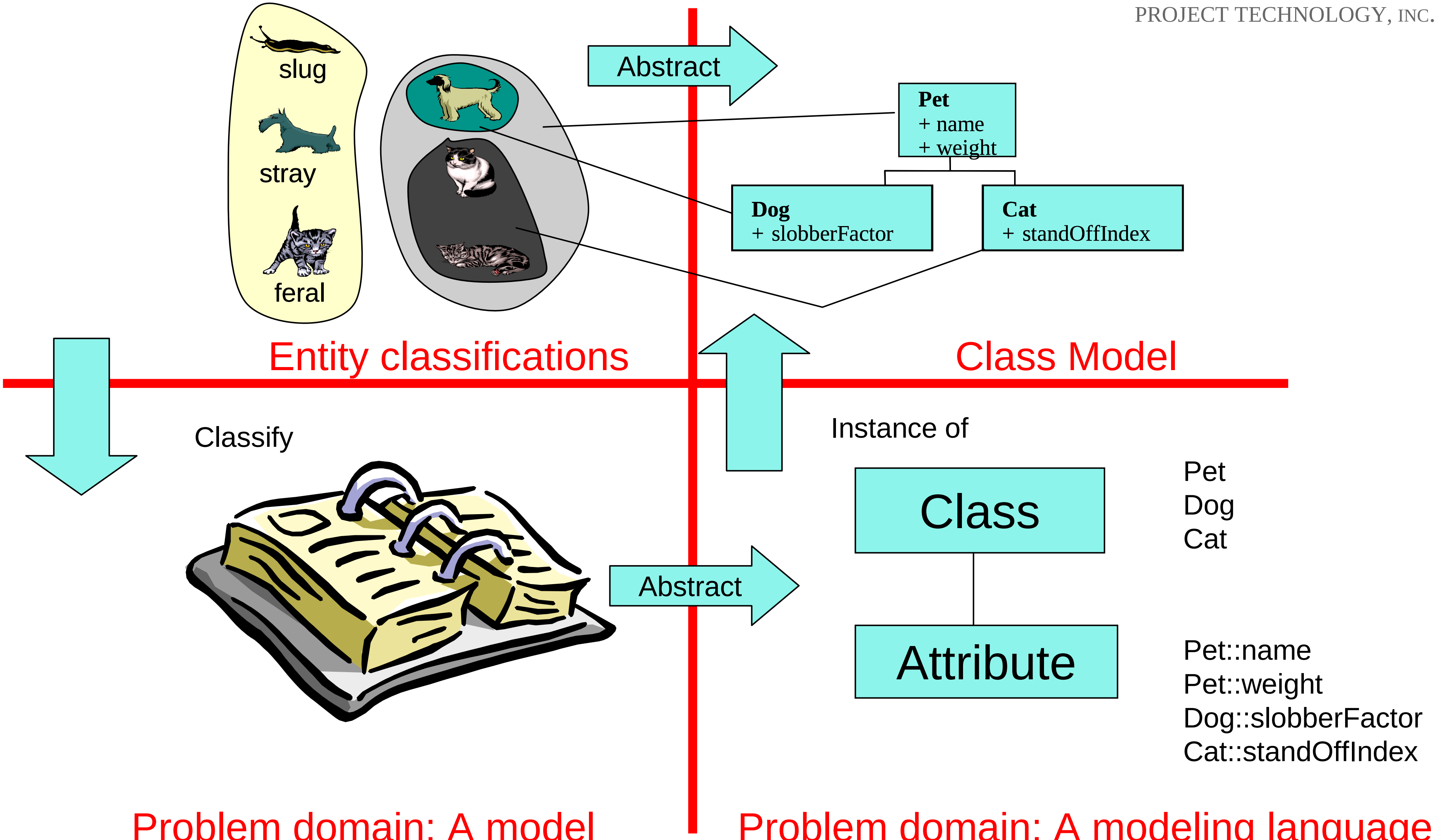
Dog
+ slobberFactor

Cat
+ standOffIndex

Problem domain

Model

The relationship to the metamodel



slug

stray

feral

Abstract

Pet
+ name
+ weight

Dog
+ slobberFactor

Cat
+ standOffIndex

Entity classifications

Class Model

Classify

Instance of

Abstract

Class

Attribute

Pet
Dog
Cat

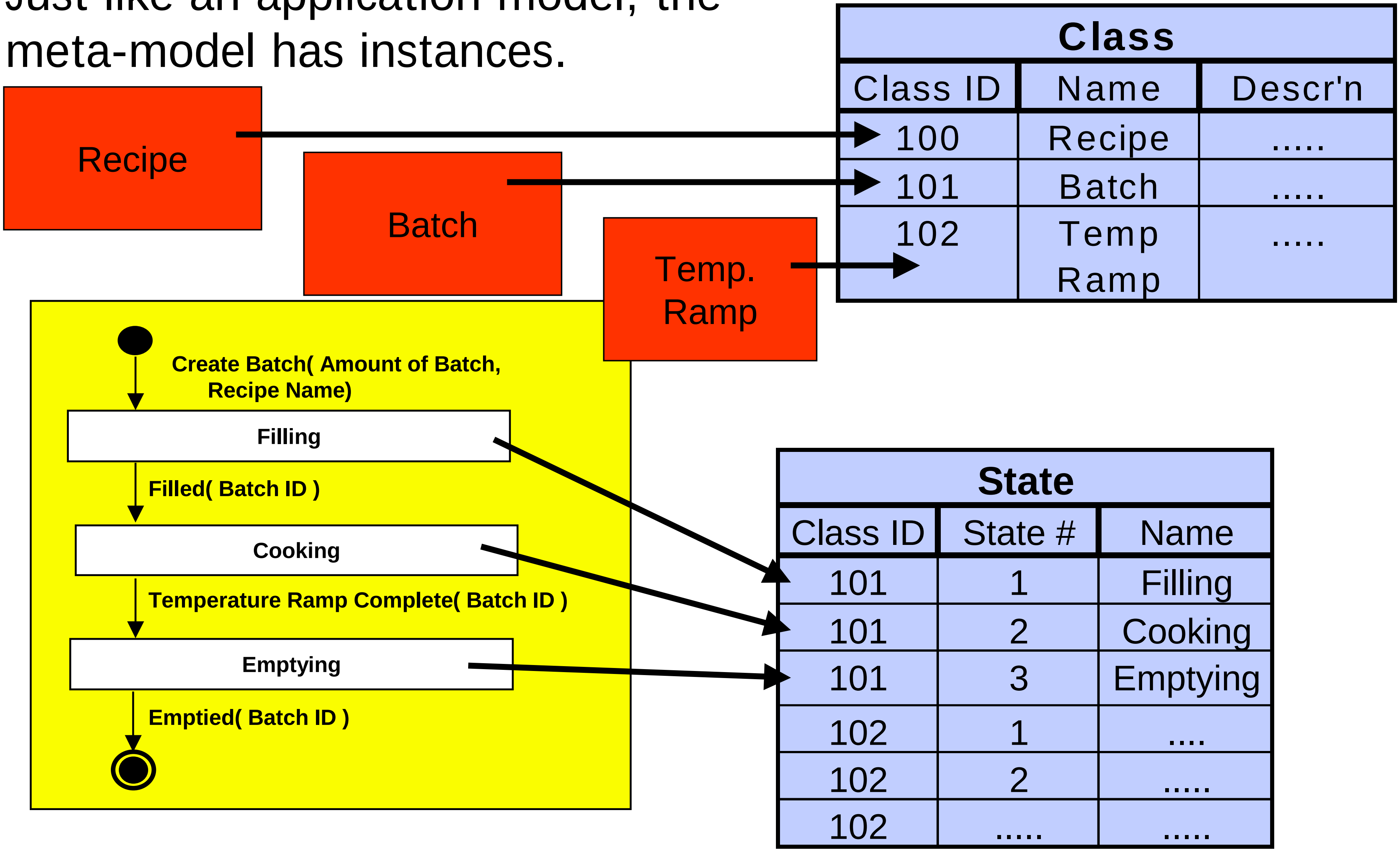
Pet::name
Pet::weight
Dog::slobberFactor
Cat::standOffIndex

Problem domain: A model

Problem domain: A modeling language
(I.e. a Metamodel)

Metamodel instances

Just like an application model, the meta-model has instances.


Recipe
Batch
Temp. Ramp
Create Batch(Amount of Batch, Recipe Name)
Filling
Filled(Batch ID)
Cooking
Temperature Ramp Complete(Batch ID)
Emptying
Emptied(Batch ID)

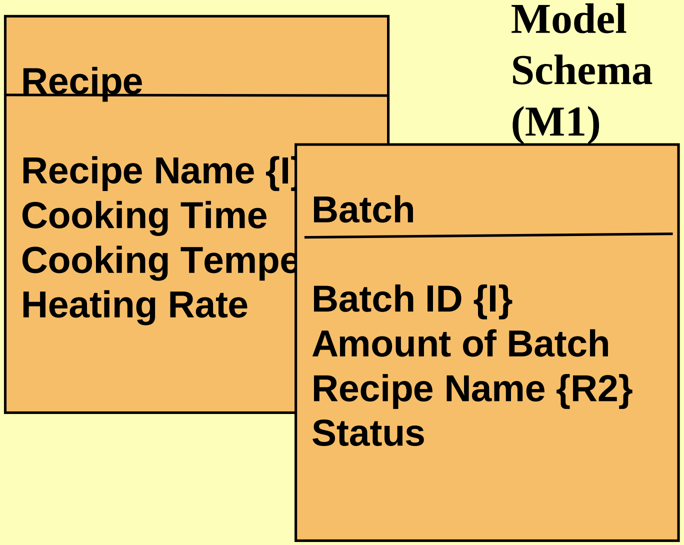
Class

Class ID	Name	Descr'n
100	Recipe	.....
101	Batch	.....
102	Temp Ramp	.....

State

Class ID	State #	Name
101	1	Filling
101	2	Cooking
101	3	Emptying
102	1	....
102	2	.....
102	.....	.....

Model Schema (M1)



Model Instances (M0)

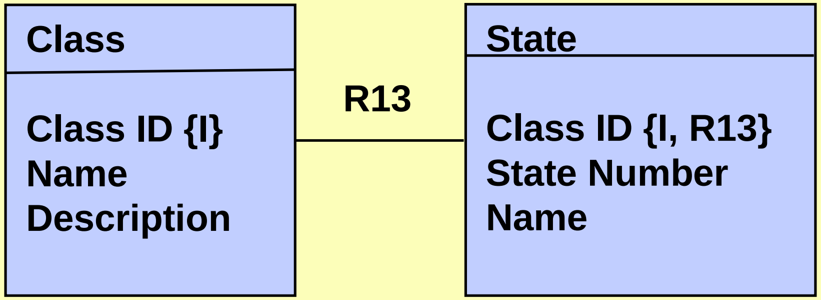
Recipe

Recipe Name	Cooking Time	Cooking Temp	Heating Rate
Nylon	23	200	2.23
Kevl			
Stuf			

Batch

Batch ID	Amount of Batch	Recipe Name	Status
1	100	Nylon	Filling
2	127	Kevlar	Emptying
3	93	Nylon	Filling
4	123	Stuff	Cooking

MetaModel Schema (M2)



MetaModel Instances (M1)

State

Class ID	State #	Name
101	1	Filling
101	2	Cooking
101	3	
102	1	
102	2	
102	....	

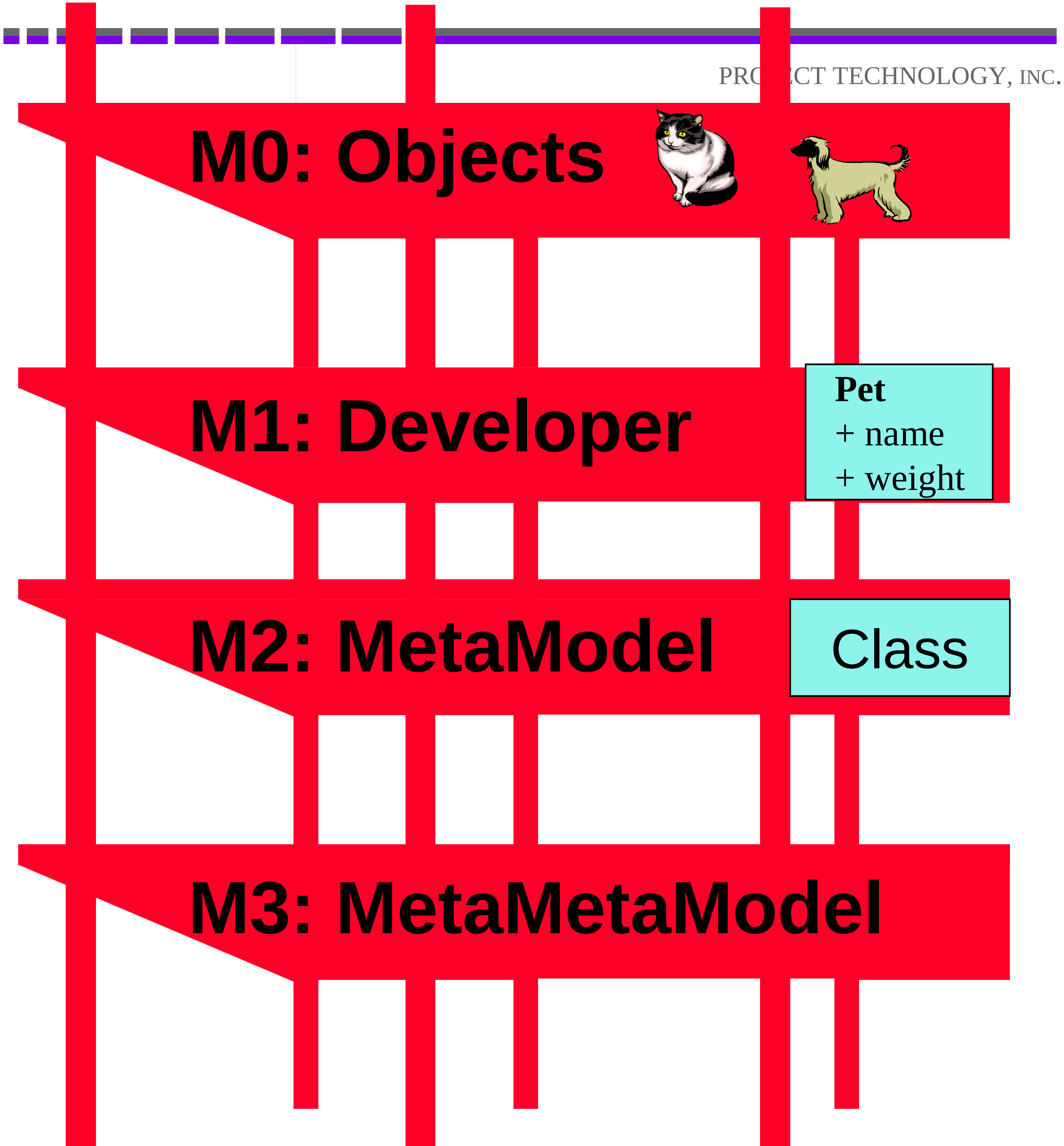
Class

Class ID	Name	Descr'n
100	Recipe	.....
101	Batch	.....
102	Temp Ramp	.....

Four-layer architecture

The “four-layer architecture” is a simple way to refer to each layer.

(In reality, meta-levels are relative.)

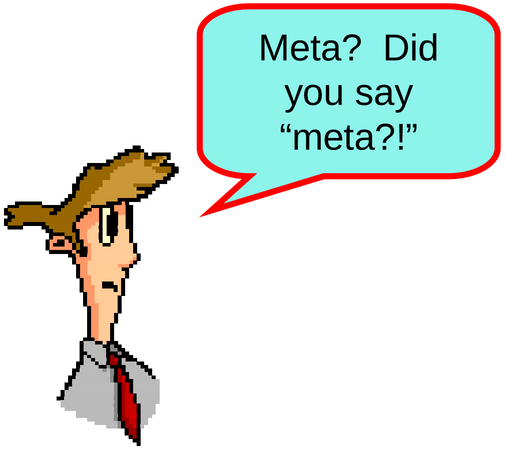


Fourth Layer

The fourth layer is a *model of the metamodel*, which yields a “meta-meta-model.” It is the simplest model that can model the metamodel.

A metamodel of the “meta-meta-model” (i.e. the “meta-meta-meta-model”) would have the same structure as the meta-meta-model. This layer is:

- Reflective
- Normally associated with the MOF



MOF

The Meta-Object Facility is an OMG standard that defines the structures for M3.

Any metamodel can be captured in MOF (not just UML), which makes it the basis

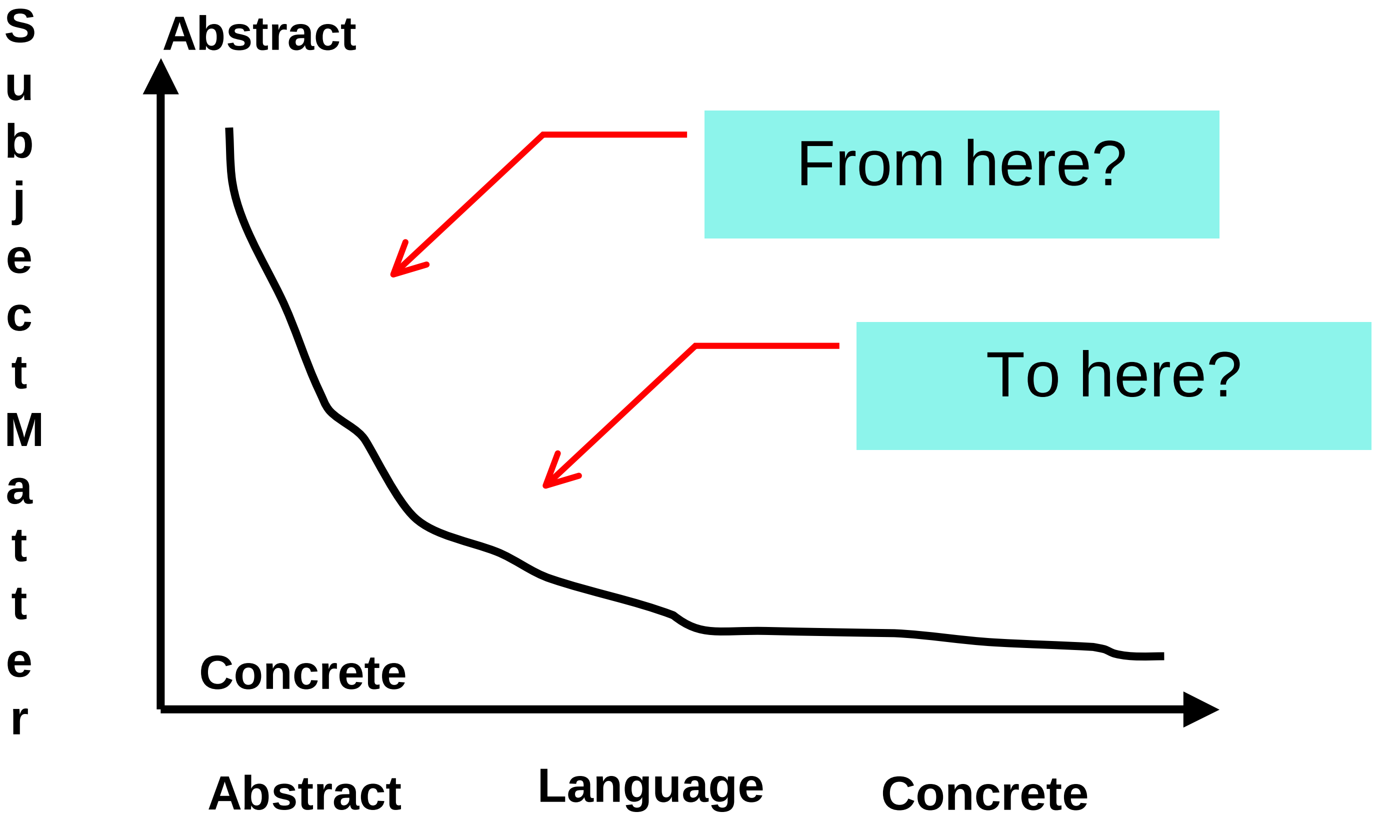
- for defining standards that ...
- *...map between metamodels*.

Table of contents

1. What’s the problem?
2. Models
3. Metamodels
4. Mappings
5. Marks
6. Building a Language
7. Agile MDA
8. Conclusion

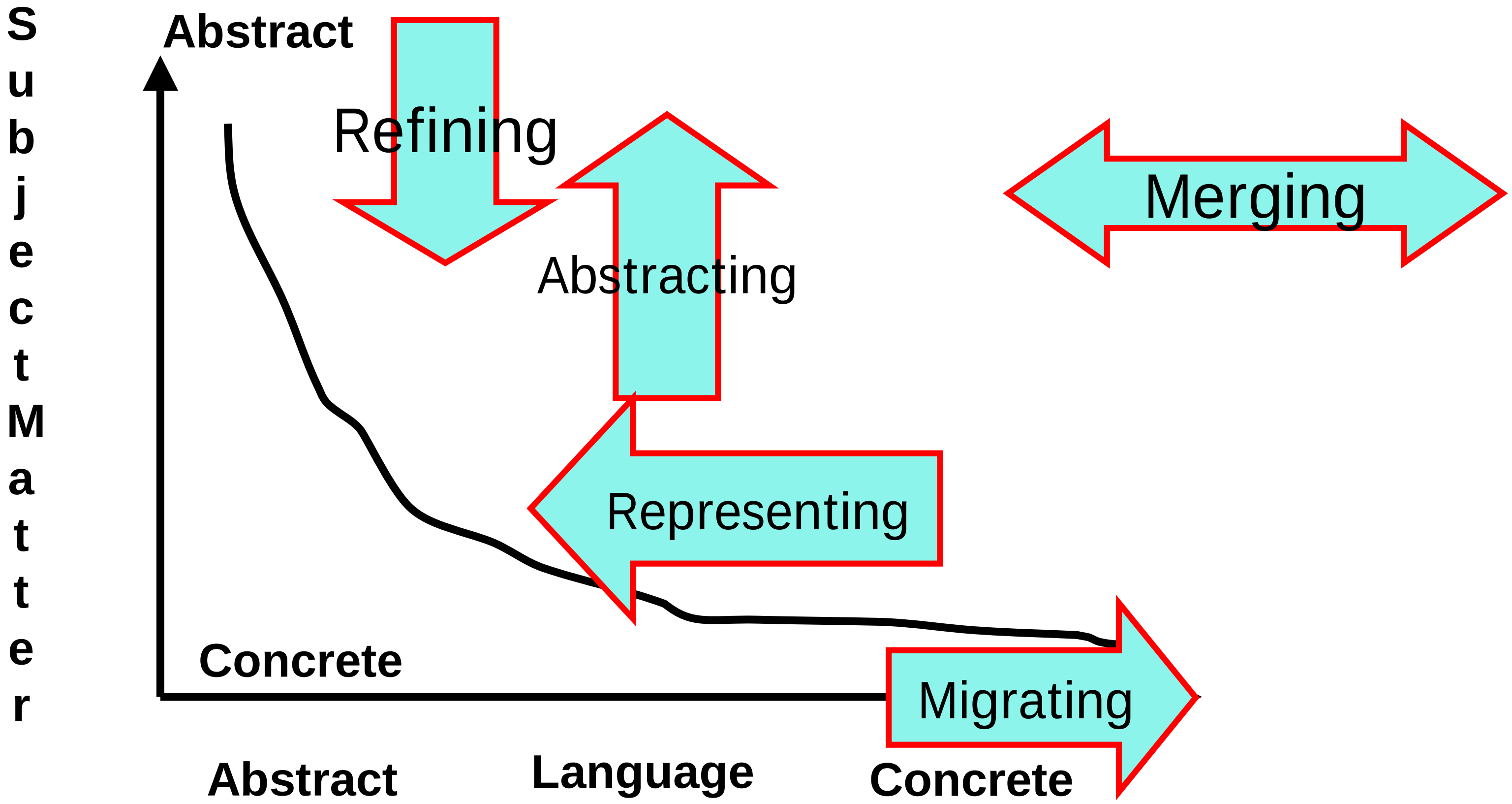
Mapping functions

A mapping function transforms one model into another.



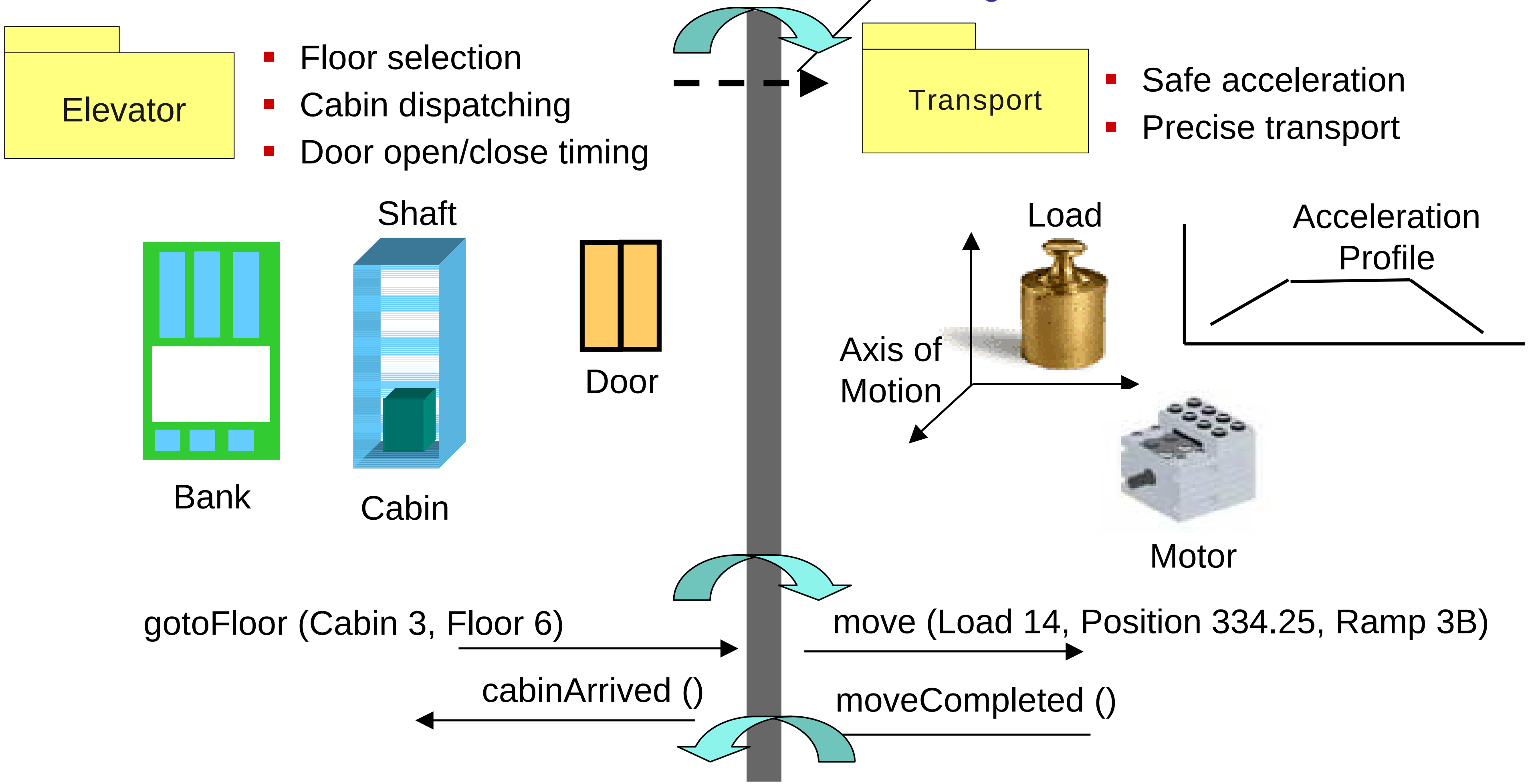
Types of mappings

In general, a mapping can be:



Example of merging mapping

Elevator uses Transport
Bridge between domains



- Floor selection
- Cabin dispatching
- Door open/close timing

- Safe acceleration
- Precise transport

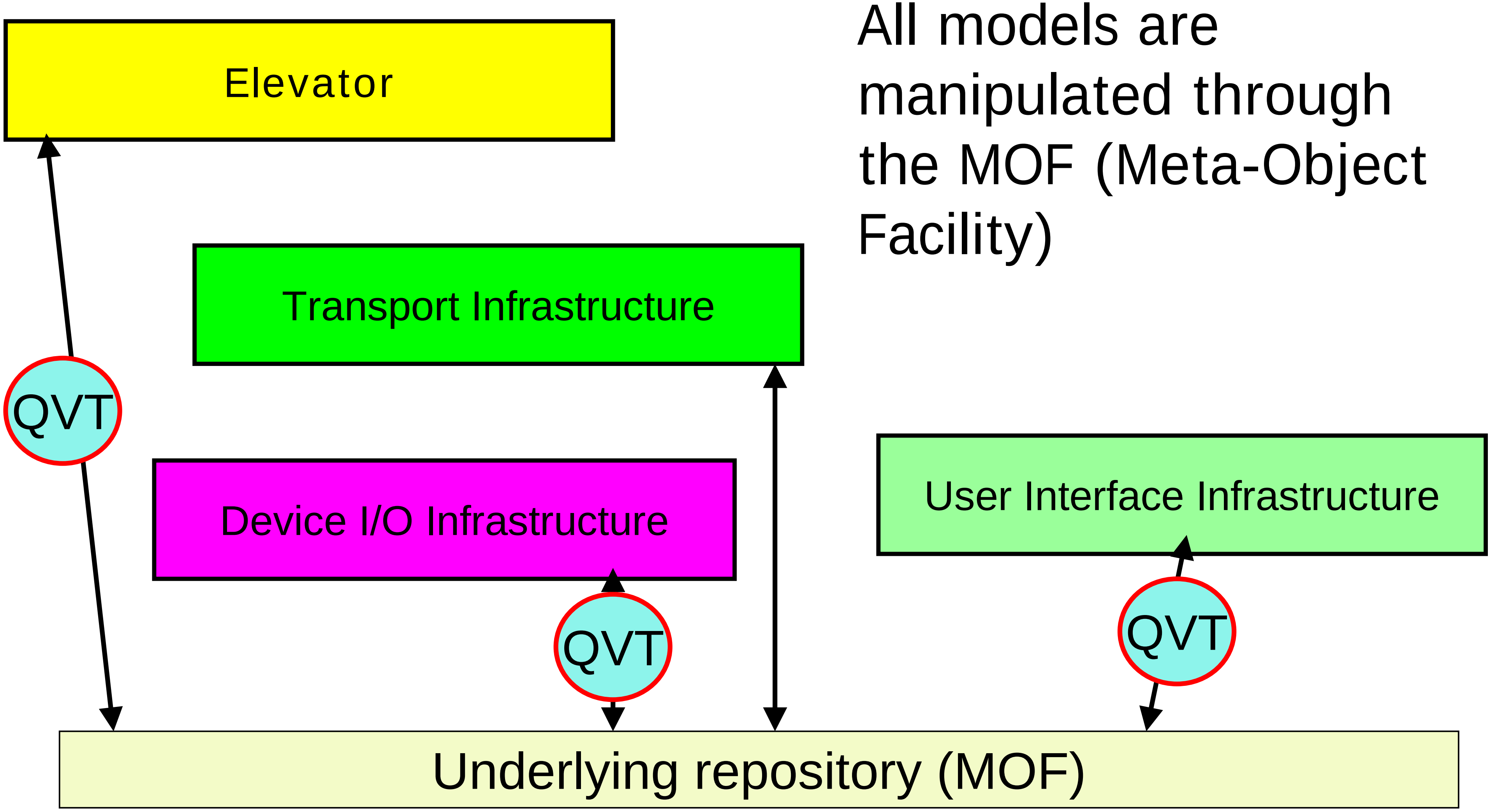
gotoFloor (Cabin 3, Floor 6)

cabinArrived ()

move (Load 14, Position 334.25, Ramp 3B)

moveCompleted ()

Metamodel-metamodel mappings


Elevator
Transport Infrastructure
Device I/O Infrastructure
User Interface Infrastructure
QVT
QVT
QVT
Underlying repository (MOF)

All models are manipulated through the MOF (Meta-Object Facility)

Why MOF?

A metamodel (as stored in MOF) allows us to state mapping rules.

- For each Class....
- For each Structural Feature...
- For each Attribute...
- For each Action

rather than manipulate specific classes in the developer model.

This means a standard “mapping tool” can be defined: QVT.

Metamodel-metamodel mappings

```
.function Transform
.param inst_ref class
.open OOA, Arch;
.select many PDMs related by
    class->attribute[R105] in OOA
.for each PDM in PDMs
Insert PDM in PDMTable in Arch;
.endfor
.end function
```

QVT is a standard approach for defining *mapping functions*

that map between metamodels

Inserts element (“attribute”) in target metamodel.

- Query
- View
- Transform

QVT

There is presently no standard, but three approaches present themselves:

- Imperative,
- Template,
- Declarative.

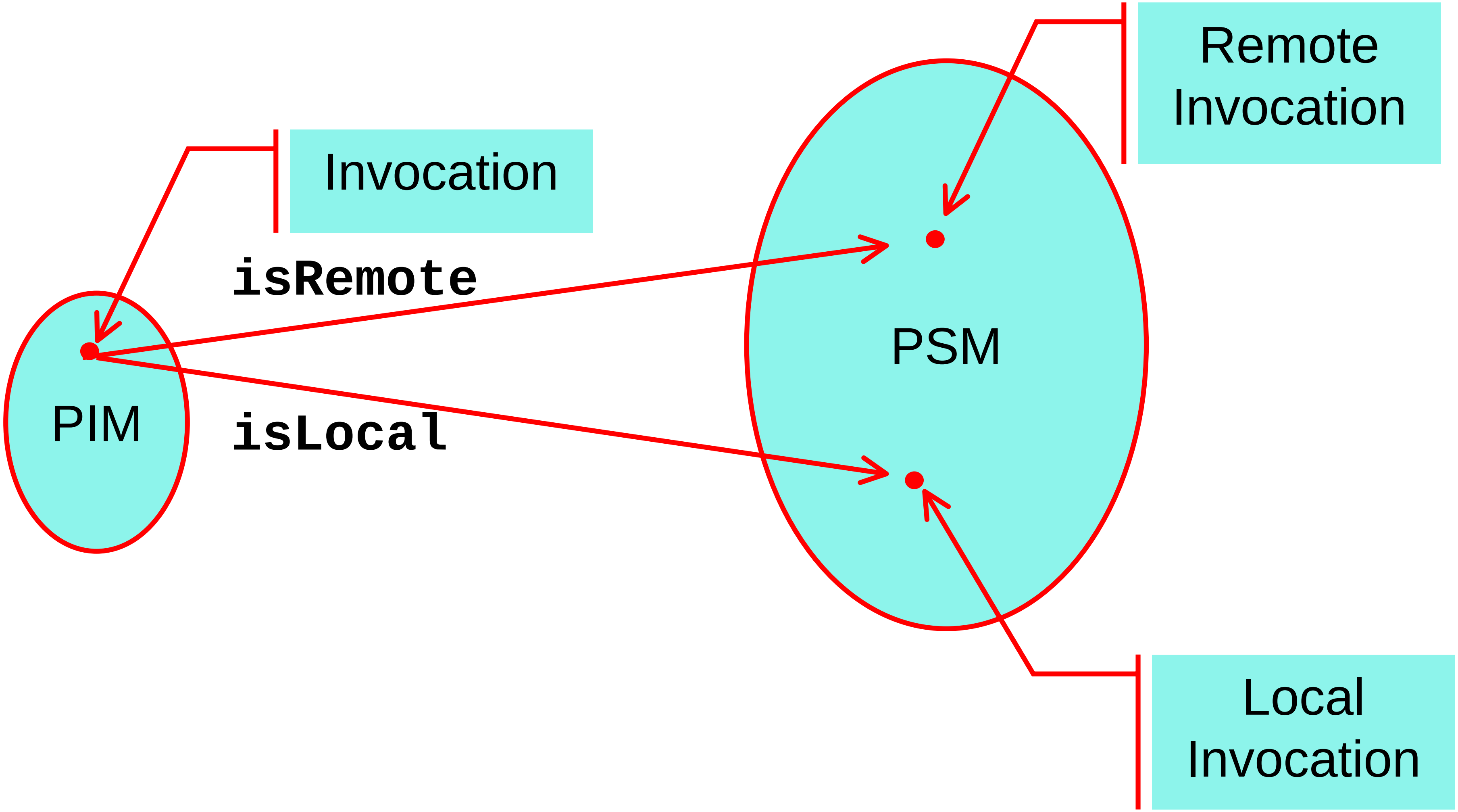
The RFP explicitly demands declarative, but alternatives have been proposed.

Table of contents

1. What’s the problem?
2. Models
3. Metamodels
4. Mappings
5. Marks
6. Building a Language
7. Agile MDA
8. Conclusion

Why marks?

A *mark* distinguishes multiple possible targets.



Applying marks

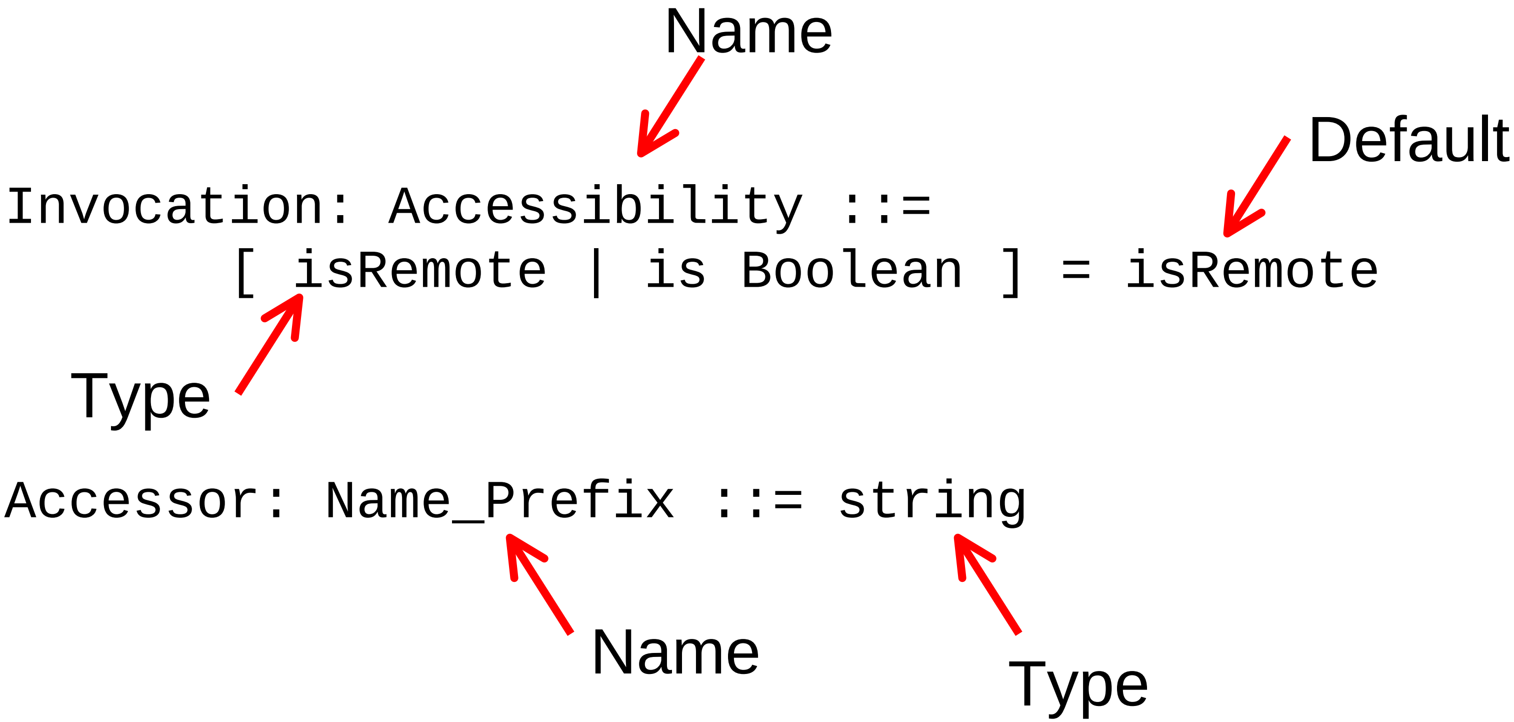
Marks may be used as:

- Rule selection:
  - If the mark has value isRemote, invoke a remote accessor, otherwise...
- Value provider:
  - Prefix the (string) value to all marked elements (E.g. add the string “db_” to all db accessors)

Marking models

A *marking model* is a way to declare:

- Names of marks
- Their types
- Defaults (if any)



```
Invocation: Accessibility ::=
       [ isRemote | is Boolean ] = isRemote

Accessor: Name_Prefix ::= string
```

Name

Default

Type

Name

Type

Relating marks to metamodel types

Marks are associated with metamodel elements.

Model element

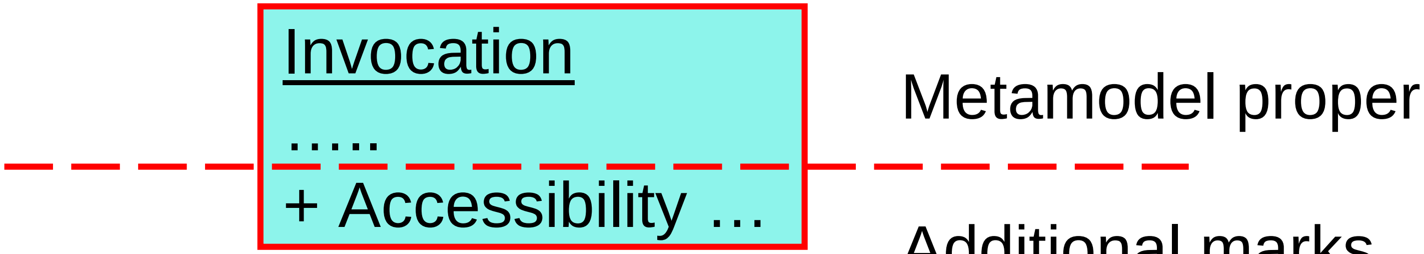
```
Invocation: Accessibility ::=
     [ isRemote | is Boolean ] = isRemote


Accessor: Name_Prefix ::= “db_” : string
```

Model element

“are associated with?”

Both *Invocation* and *Accessor* are UML metamodel elements.

The marks *Accessibility* and *Name_prefix* describe these metamodel elements, but are ***not*** a part of them.



Other marks

Some marks are “constants.”

- For example, a postfix to all class names

You can think of these as marks that apply to the *meta*model (M2)

Some marks apply to instances

- For example, processor allocation for fixed-input devices

You can think of these as marks that apply to the *instance* model (M1).

Theory of marks

There isn’t one. Yet.

But:

- What should be parameterized as a function vs. a mark?
- Can there be a taxonomy of marks?
- What are good/bad ways to use marks?
- Should marks be prescriptive, or should they describe the source model and let the mapping function decide?
  - For example, is it better to say “linked list” or say “few instances,” which *might* then imply a linked list?

Table of contents

1. What’s the problem?
2. Models
3. Metamodels
4. Mappings
5. Marks
6. Building a Language
7. Agile MDA
8. Conclusion

Languages

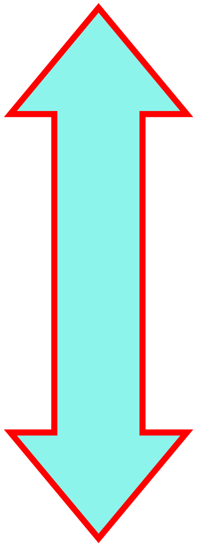
We build languages all the time.

- When we subset the UML for our preferred elements
- When we extend it by adding adornments or notes

We must decide if we need to do so formally.

Language definers include:

- Standards bodies
- Tool vendors
- Methodology definers
- MDA architects
- Developers



Increasing

Need for formality

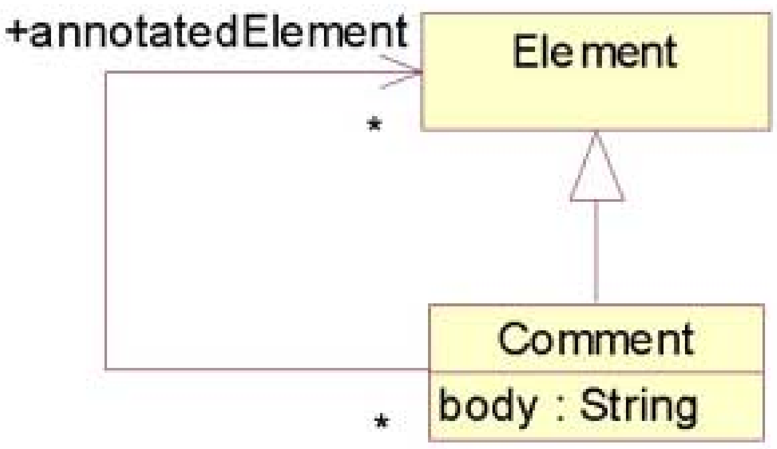
Decreasing

Building a language using MOF

MOF is an (object-oriented) metamodeling language, so:

- It can be used to create a language.
  - For example, UML

You can use MOF to create your own modeling language



Building a language using profiles

A *profile* is a UML mechanism used to define and extend metamodels.

- Profiles may be used to define metamodels for PIMs and PSMs
- Profiles may be used to define marking models

A profile is defined in terms of:

- *Stereotypes* that extend “meta-”classes, and
- *Constraints*, defined using OCL

Example

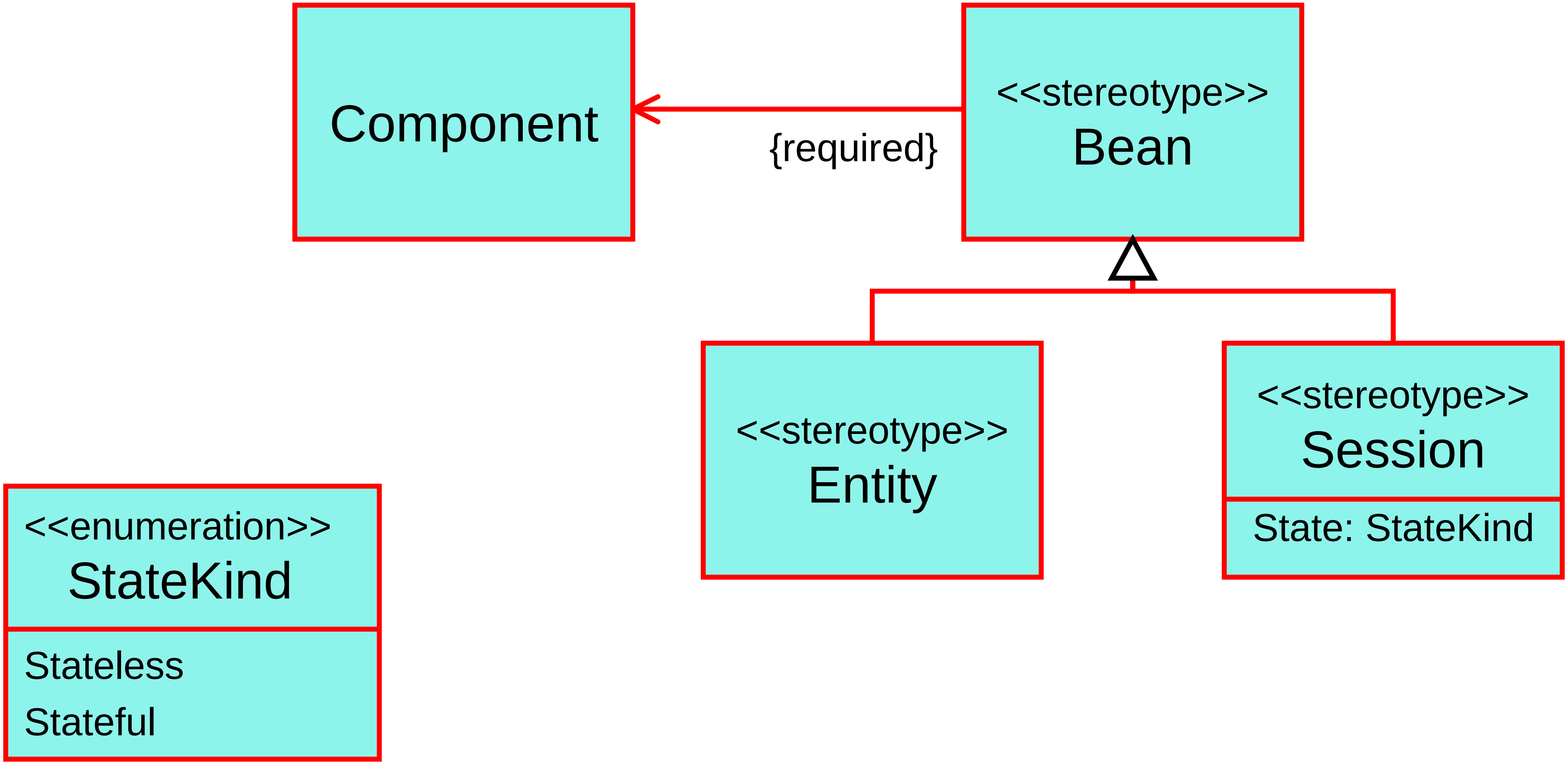
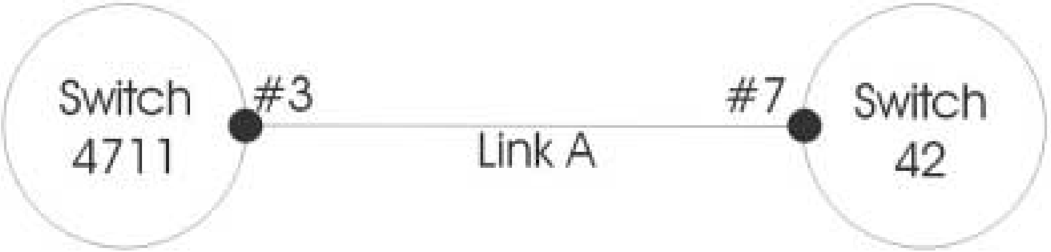


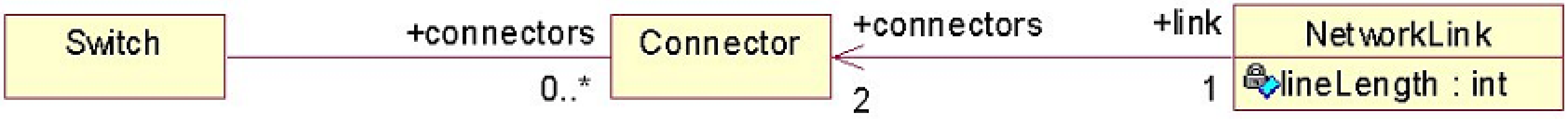
Figure 12-99: A simple EJB profile
Superstructure submission

Building graphical notation (for a language)

In a networking problem, we may want to draw:



which may be captured as:



with instances:

- 4711
- 42

- 3, 4711
- 7, 42

- A, 3, 7, 173

line
length

Building graphical notation (for a language)

By mapping the model to UML, we get drawing tools for “free.”

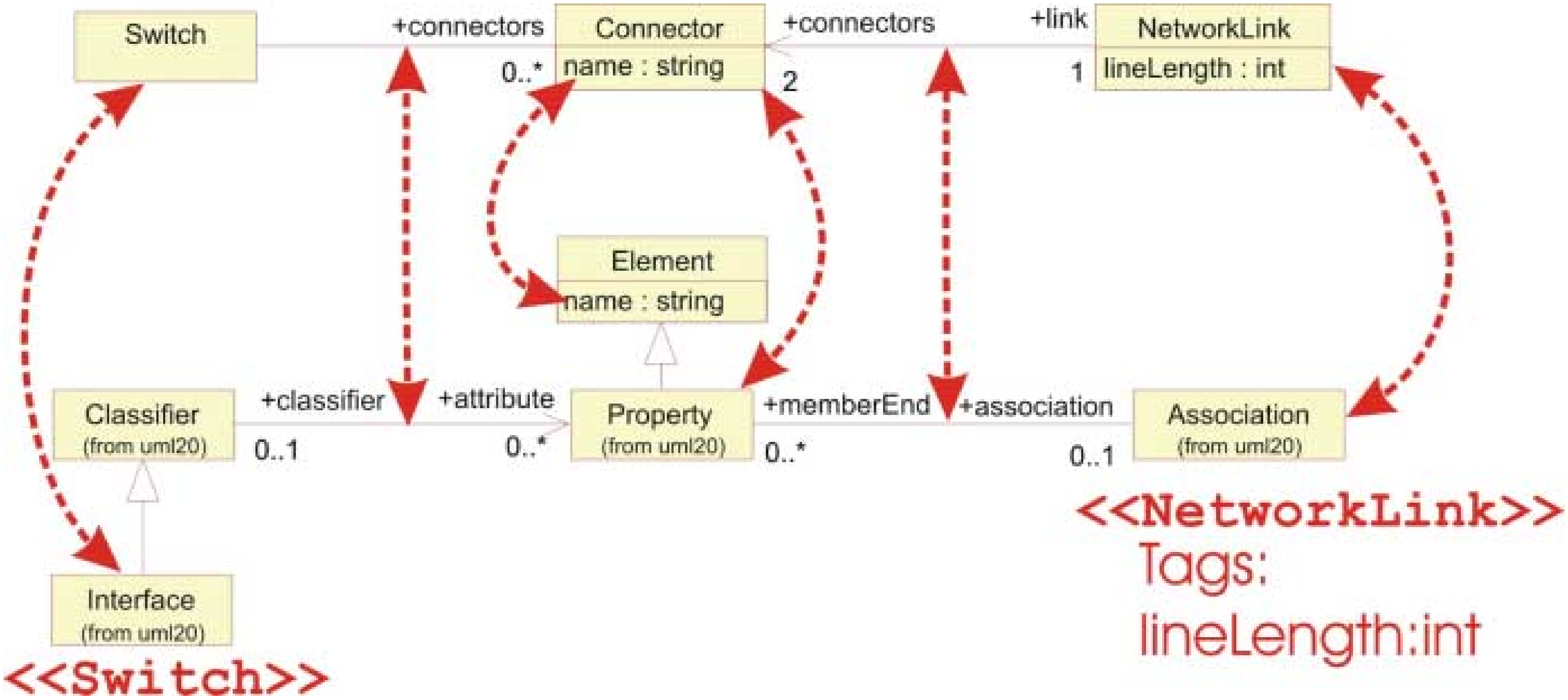


Table of contents

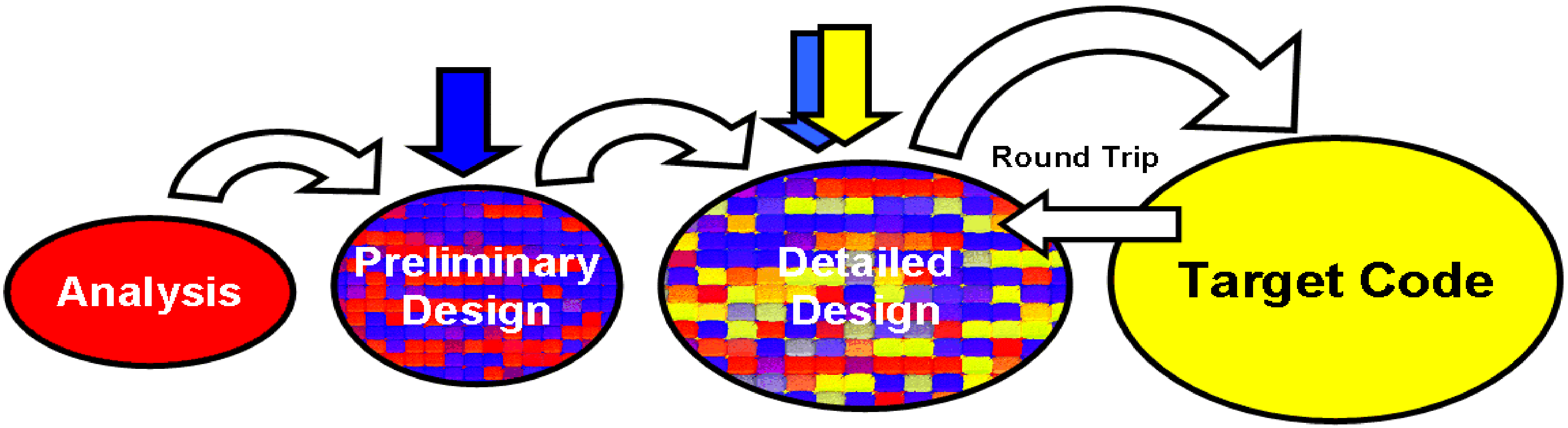
1. What’s the problem?
2. Models
3. Metamodels
4. Mappings
5. Marks
6. Building a Language
7. Agile MDA
8. Conclusion

Elaborative development

Design and Architecture Details

Implementation Details and Code Bodies

Code Generation



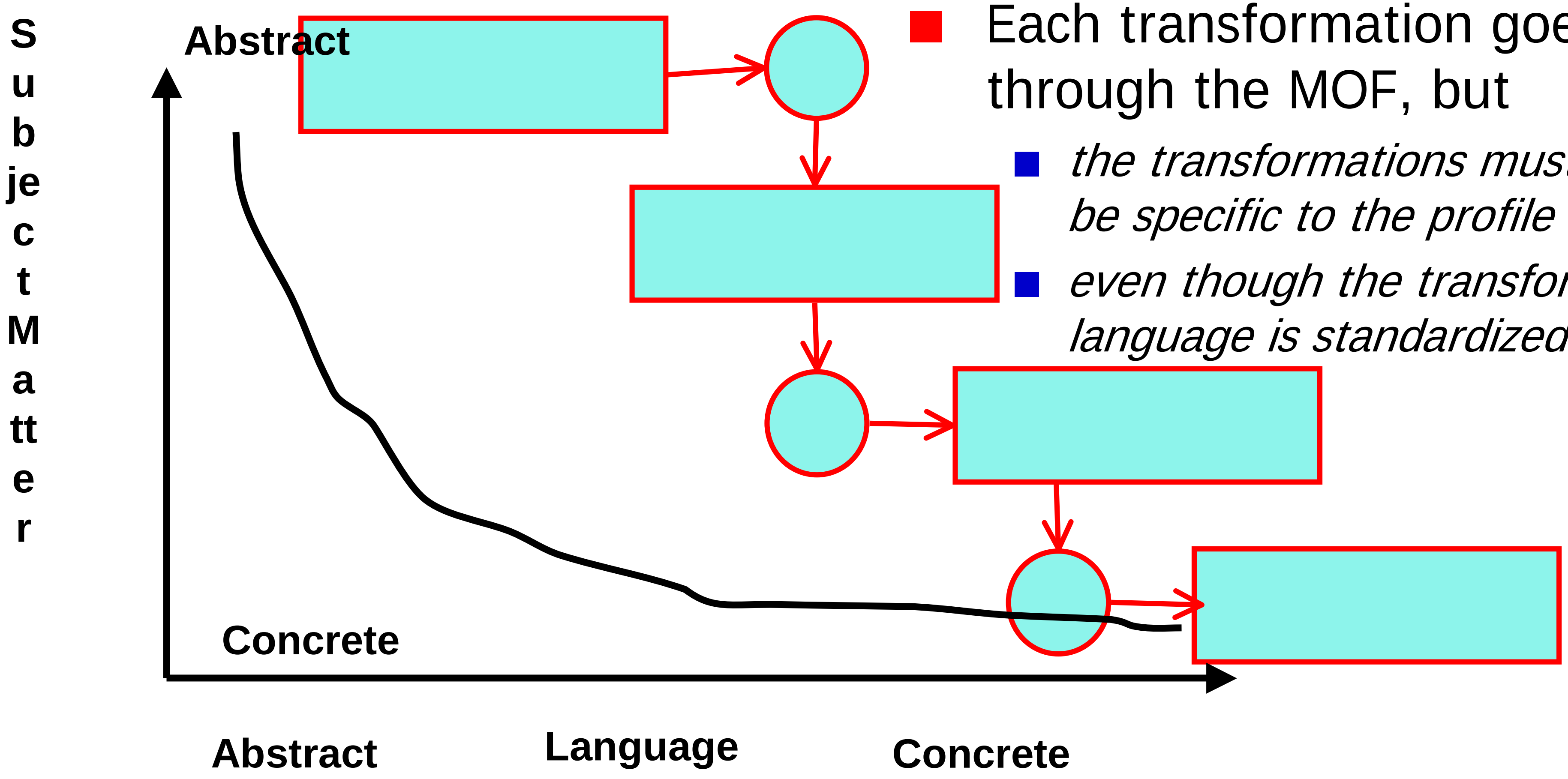
Intermixed Application and Design

Manually Created Code Bodies and Implementation Details Required for Model Execution and Code Generation

Target Code assembled from Hand-Coded Bodies inserted into a generated framework

What’s wrong with that?

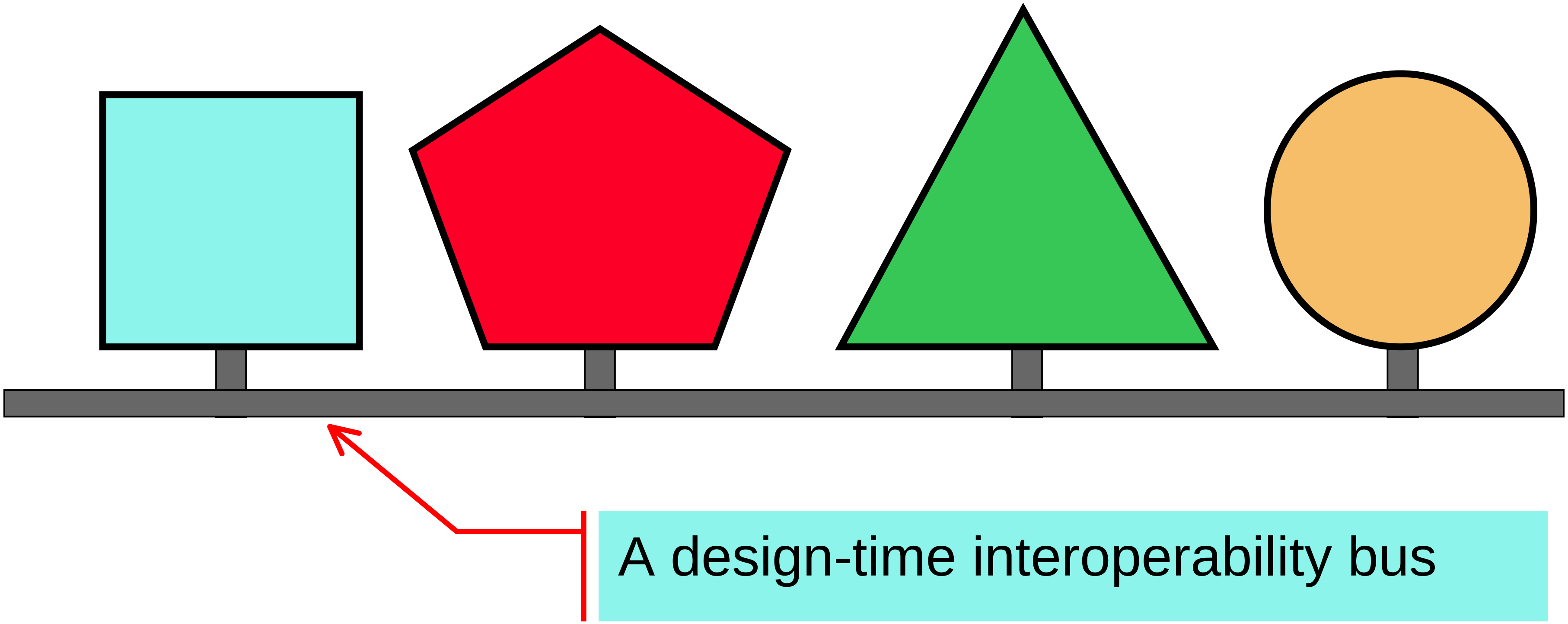
- Each meta-model demands its own profile.
- Each transformation goes through the MOF, but
  - *the transformations must be specific to the profile*
  - *even though the transformation language is standardized*



What’s the solution?

Model each domain using a:

- single neutral formalism that
- (perforce) conforms to the same metamodel

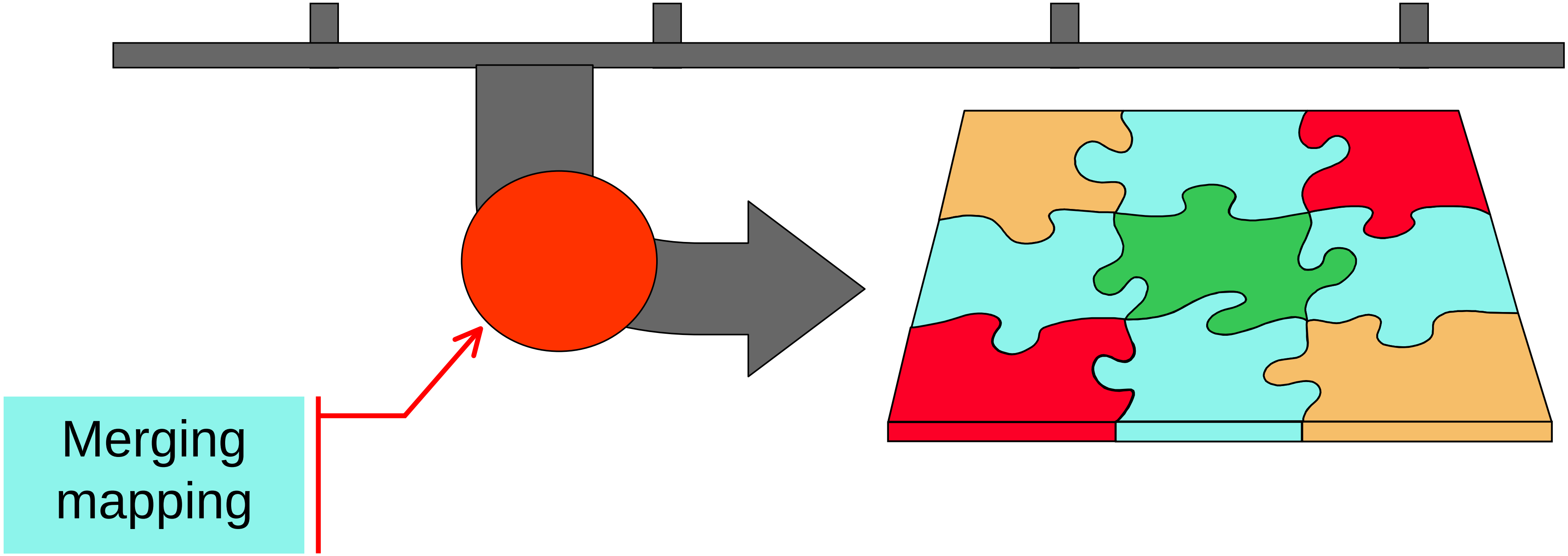


A design-time interoperability bus

What’s the solution?

Connect up the models according to:

- a single set of mapping rules that
- operate on to the same metamodel



Metamodel-to-text mappings

MDA needs a way to map data from a metamodel into text.

```
.function PrivateDataMember
.param inst_ref class
.select many PDMs related by
    class->attribute[R105]
.for each PDM in PDMs
${PDM.Type} ${PDM.Name};
.endfor
.end function
```

```
.function ClassDef
.param inst_ref class
class ${class.name} :
    public ActiveInstance {
      private:
        .invoke PrivateDataMember( class )
}
…
.end function
```

We call them “archetypes”.

Example

The archetype language produces text.

```
.select many stateS related to instances of
       class->[R13]StateChart ->[R14]State
           where (selected.isFinal == FALSE)
public:
  enum states_e
    { NO_STATE = 0 ,
.for each state in stateS
       .if ( not last stateS )
           ${state.Name } ,
       .else
           NUM_STATES = ${state.Name}
       .endif
.endfor
};
```

```
public:
  enum states_e
    {  NO_STATE = 0 ,
       Filling ,
       Cooking ,
       NUM_STATES = Emptying
    };
```

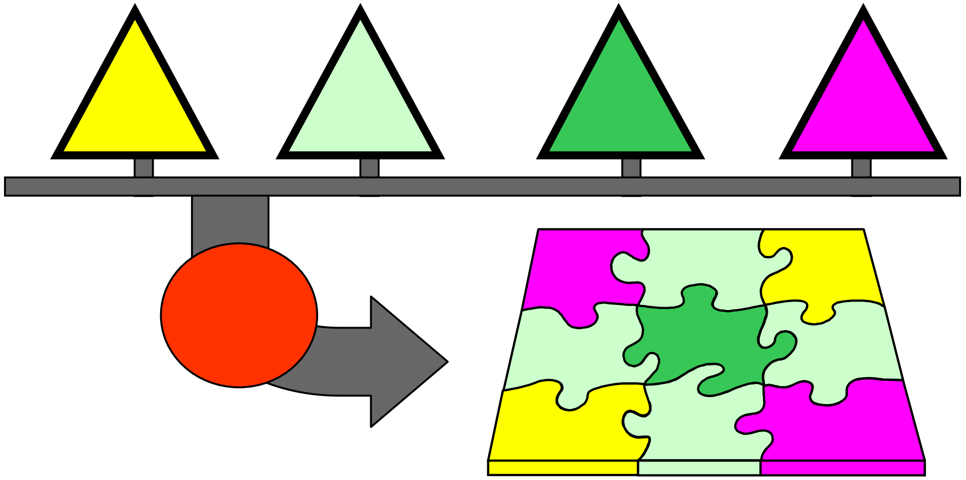
Agile MDA

- Each model we build covers a single subject matter.
- We uses the same *executable* modeling language for all subject matters.
- The executable model does not imply an implementation.
- Compose the models automatically.

This last is *design-time composability—a bus.*

Model compilers

A model compiler compiles each model according to a single set of architectural rules so that the various subject matters *are known to fit together*.



A design-time interoperability bus

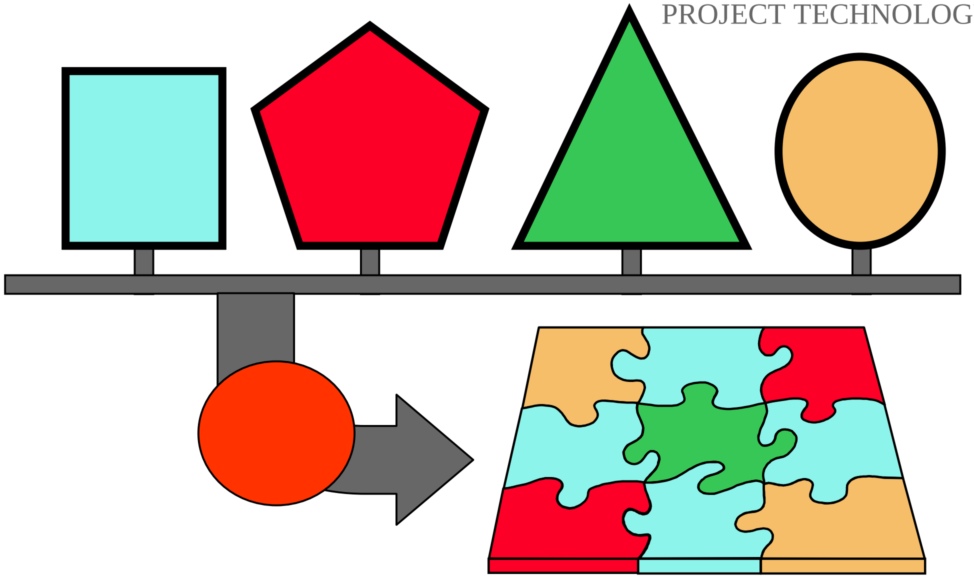
A model compiler

- Normalizes models to the infrastructure
- Combines models at design time.

Model compilers

System dimensions include:

- Concurrency and sequentialization
- Multi-processing & multi-tasking
- Persistence
- Data structure choices
- Data organization choices



● = model compiler

Examples

Financial system

- Highly distributed
- Concurrent
- Transaction-safe with rollback
- Persistence, with rollback
- C++

Embedded system

- Single task
- No operating system
- Optimized data access and storage
- C

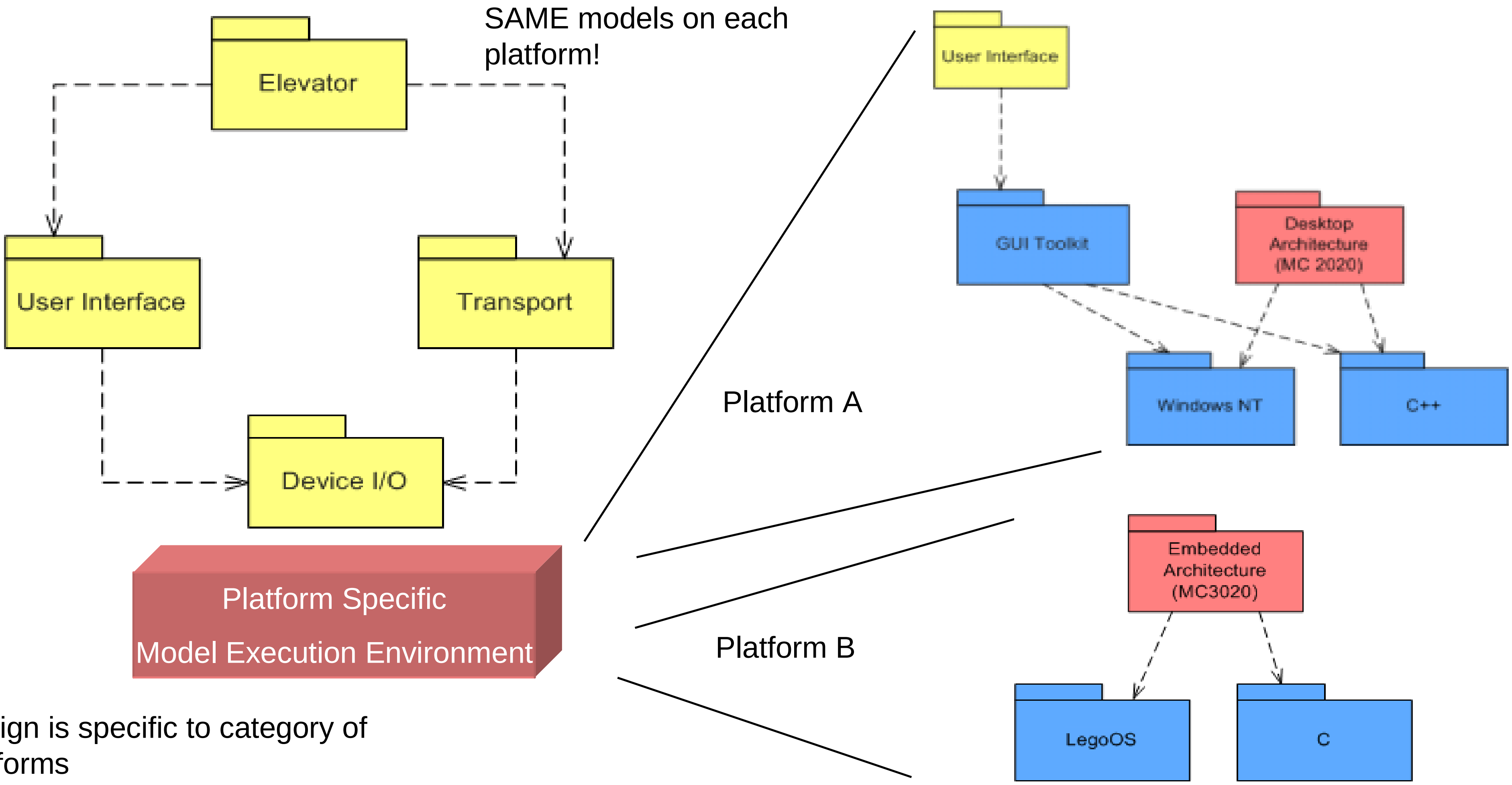
Telecommunication system

- Highly distributed
- Asynchronous
- Limited persistence capability
- C++

Simulation system

- Mostly synchronous
- Few tasks
- Special-purpose language: “Import”

All domains are translated



SAME models on each platform!

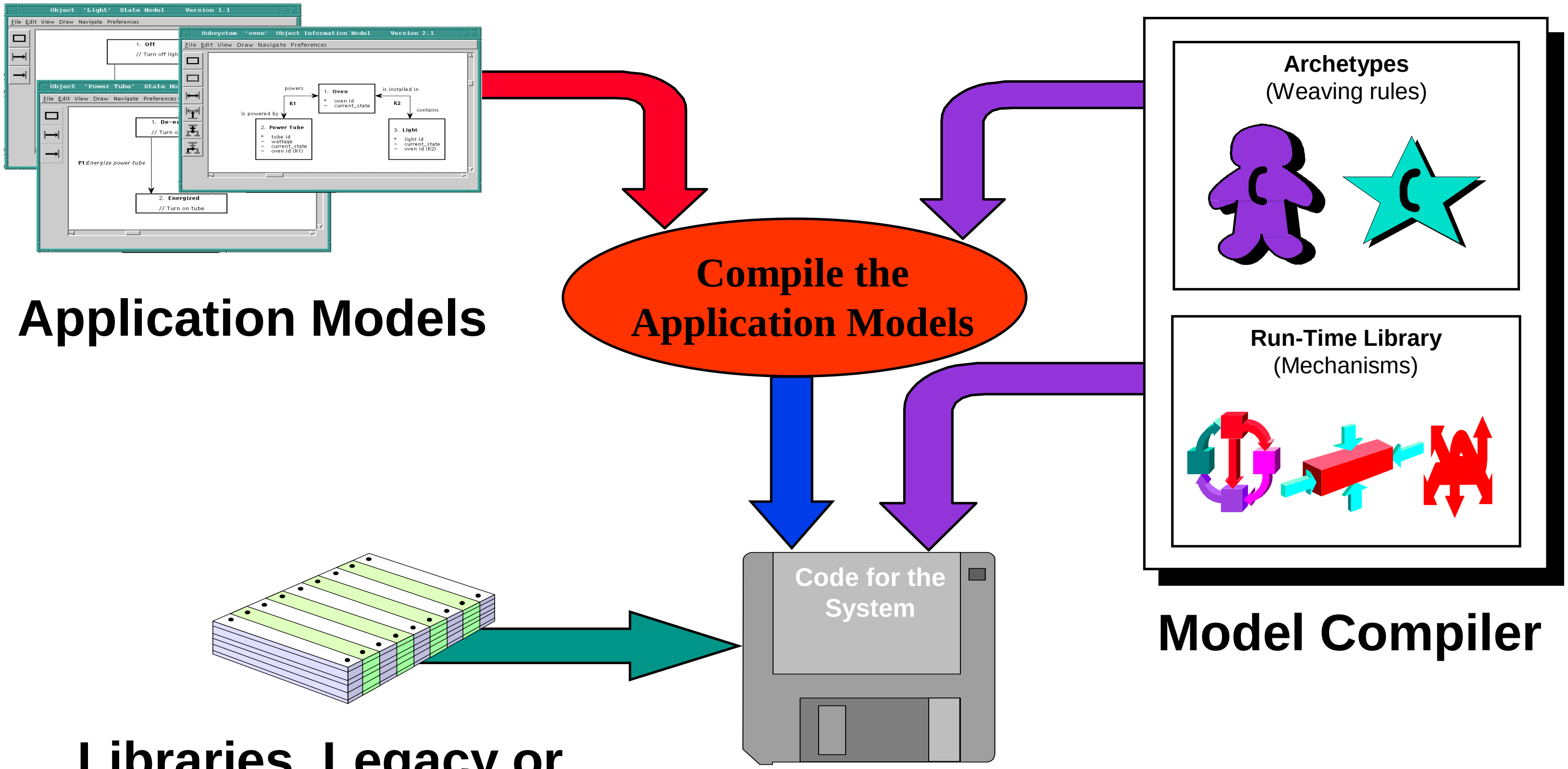
Platform A

Platform B

Design is specific to category of platforms

Building the system

Generate deliverable production code.



Application Models

Compile the Application Models

Archetypes
(Weaving rules)

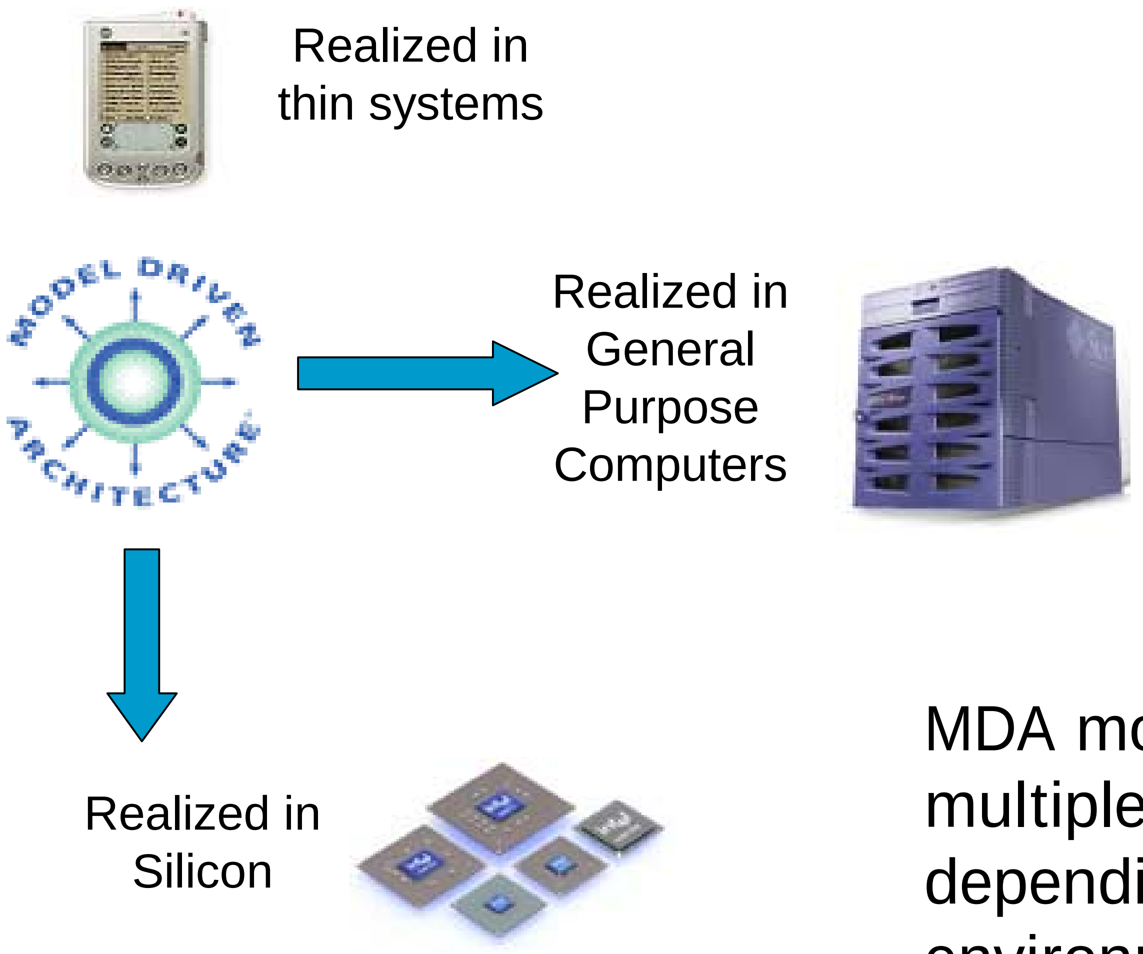
Run-Time Library
(Mechanisms)

Model Compiler

Code for the System

Libraries, Legacy or Hand-written code

Retargeting the environment



Realized in thin systems

Realized in General Purpose Computers

Realized in Silicon

MDA models can have multiple implementations depending on the target environment.

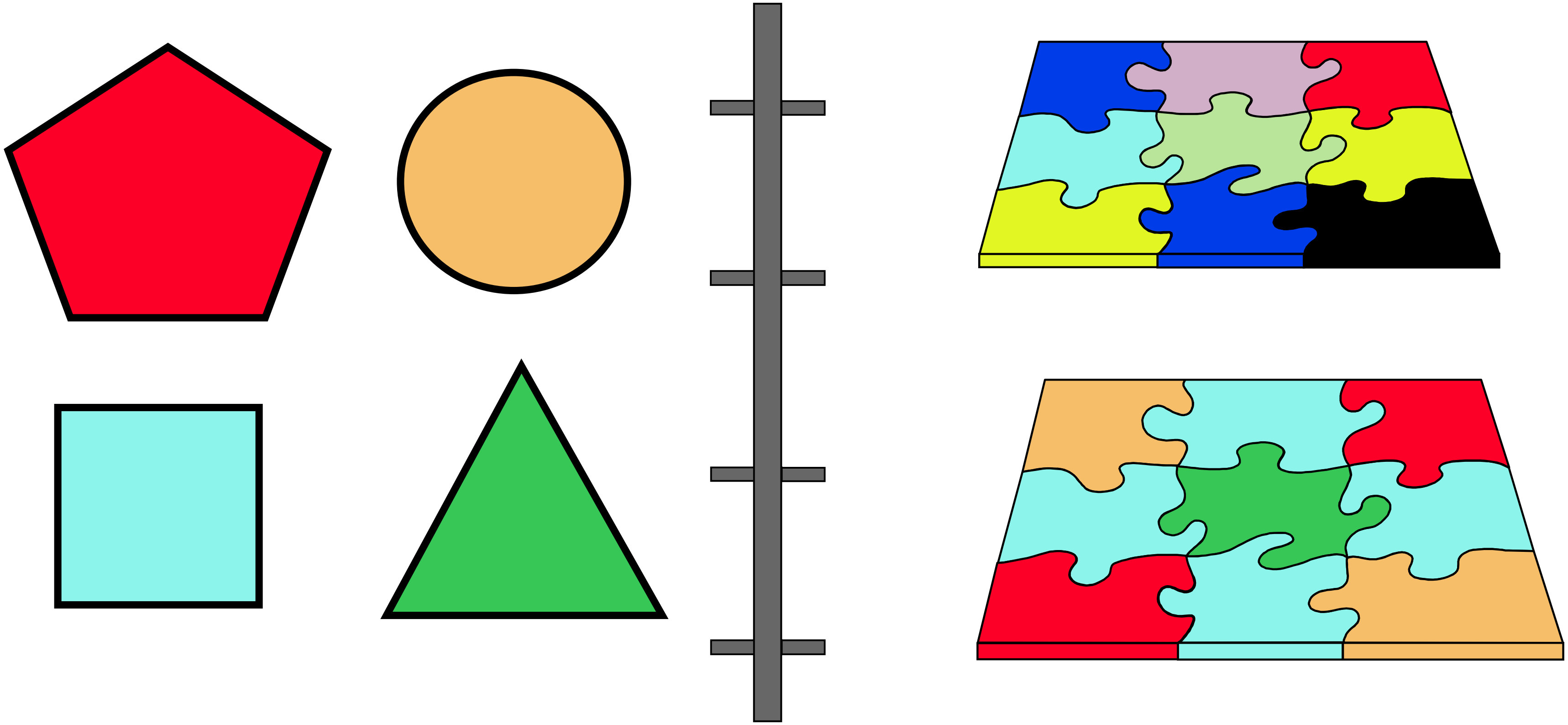
Table of contents

1. What’s the problem?
2. Models
3. Metamodels
4. Mappings
5. Marks
6. Building a Language
7. Agile MDA
8. Conclusion

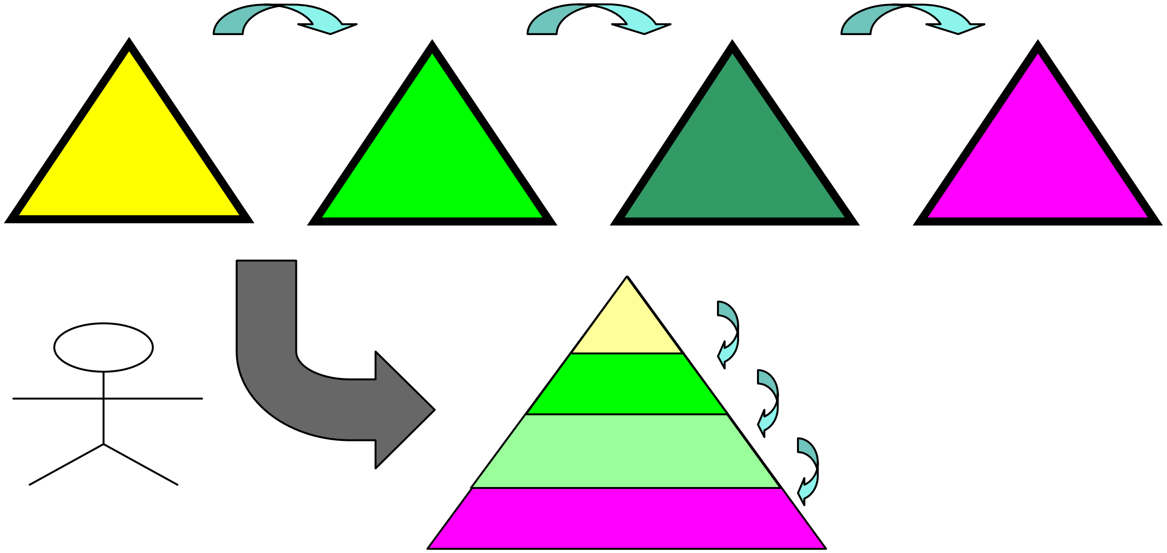
Building a market

Design time composability:

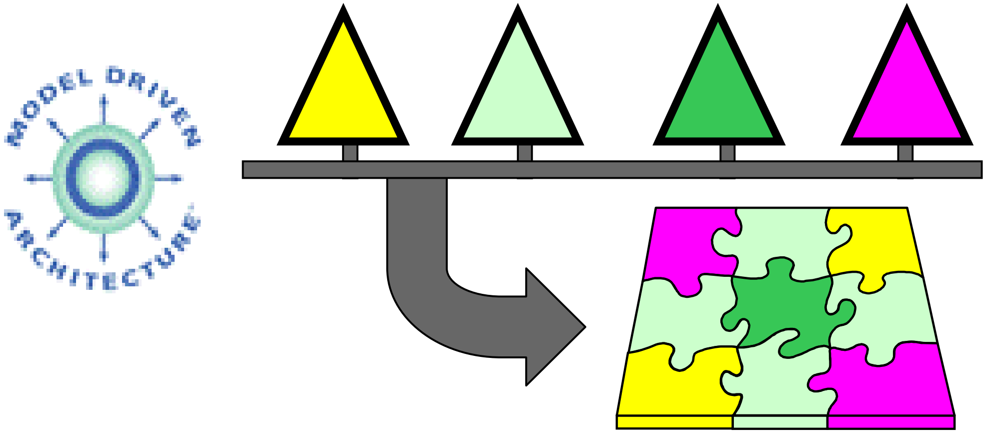
- protects IP
- allows IP to be mapped to multiple implementations
- enables a market in IP in software



MDA enables a market for IP in software!



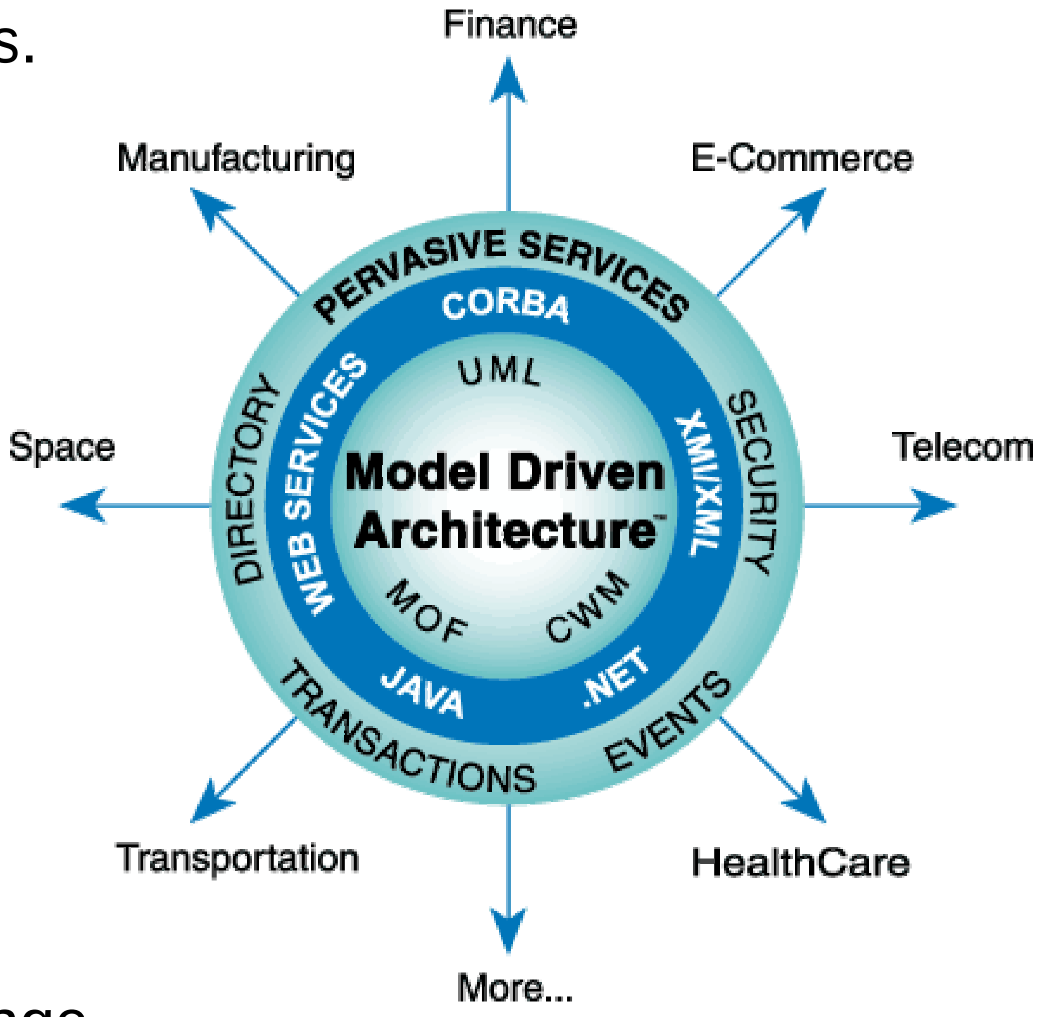
Code-driven development produces expenses.



Model-driven development produces assets.

OMG TLAs

- MOF = Meta-Object Facility a repository for metamodels.
- CWM = Common Warehouse Metamodel, which can map between models
- QVT = Query/View/ Transform, a standard for mapping between (MOF) metamodels
  - This is presently an RFP (request for proposal), and not yet a standard
- XMI = XML Model Interchange



MDA standardization

UML 2.0 Infrastructure	Jan 2003
QVT (metamodel-metamodel)	Mar 2004
Marks	Understood
Action Language	Necessary?
Archetypes (metamodel-text)	Not yet

The ADTF and the MDA WG proposes these RFPs.

See also

MDA Distilled, Mellor, Scott, Uhl and Weise
Addison-Wesley, 2004

Executable UML, Mellor and Balcer,
Addison-Wesley, 2002

www.omg.org

www.projtech.com

PROJECT TECHNOLOGY INC.

Accelerating development of high-quality systems.

Makers of BridgePoint ® Modeling Tools

Stephen J. Mellor
steve@projtech.com
Project Technology, Inc.
http://www.projtech.com