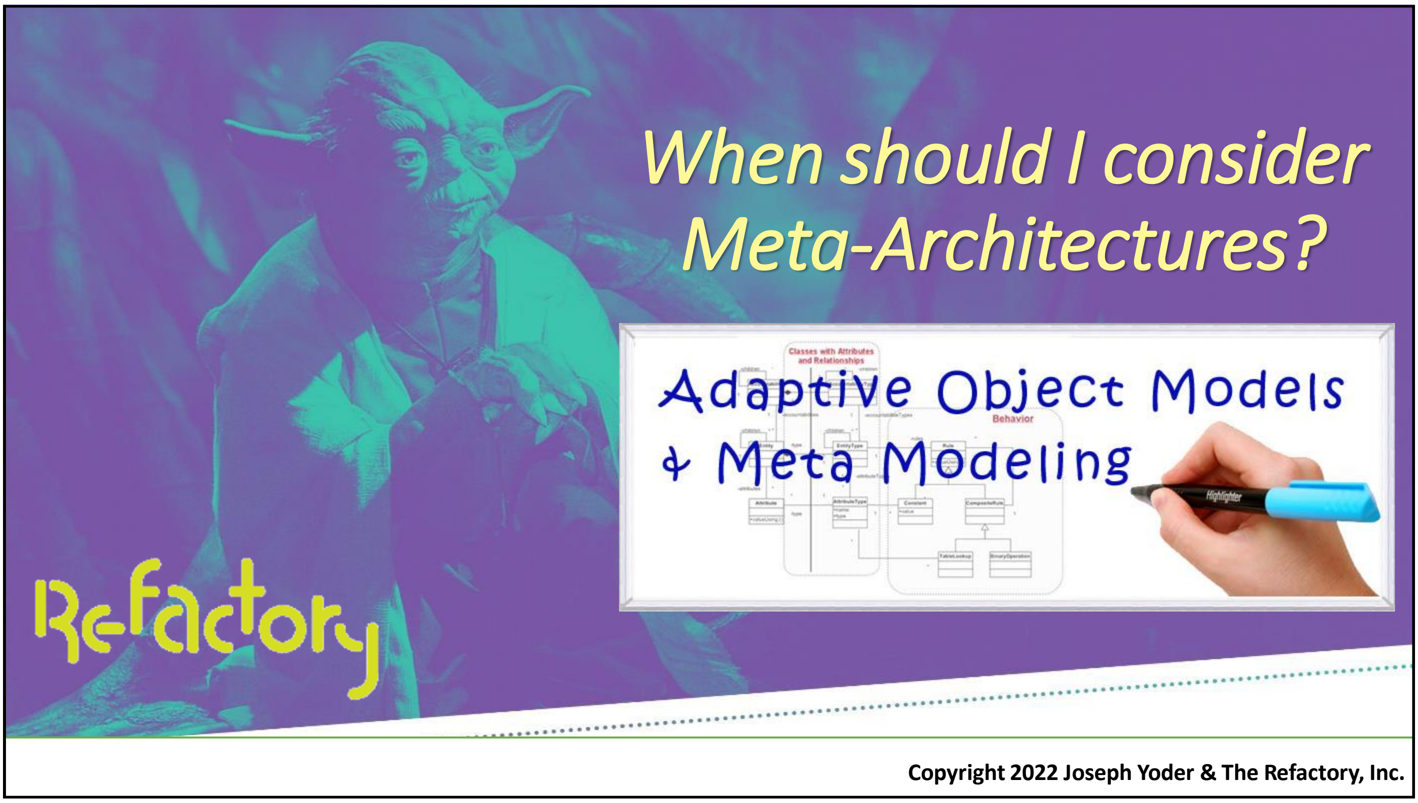
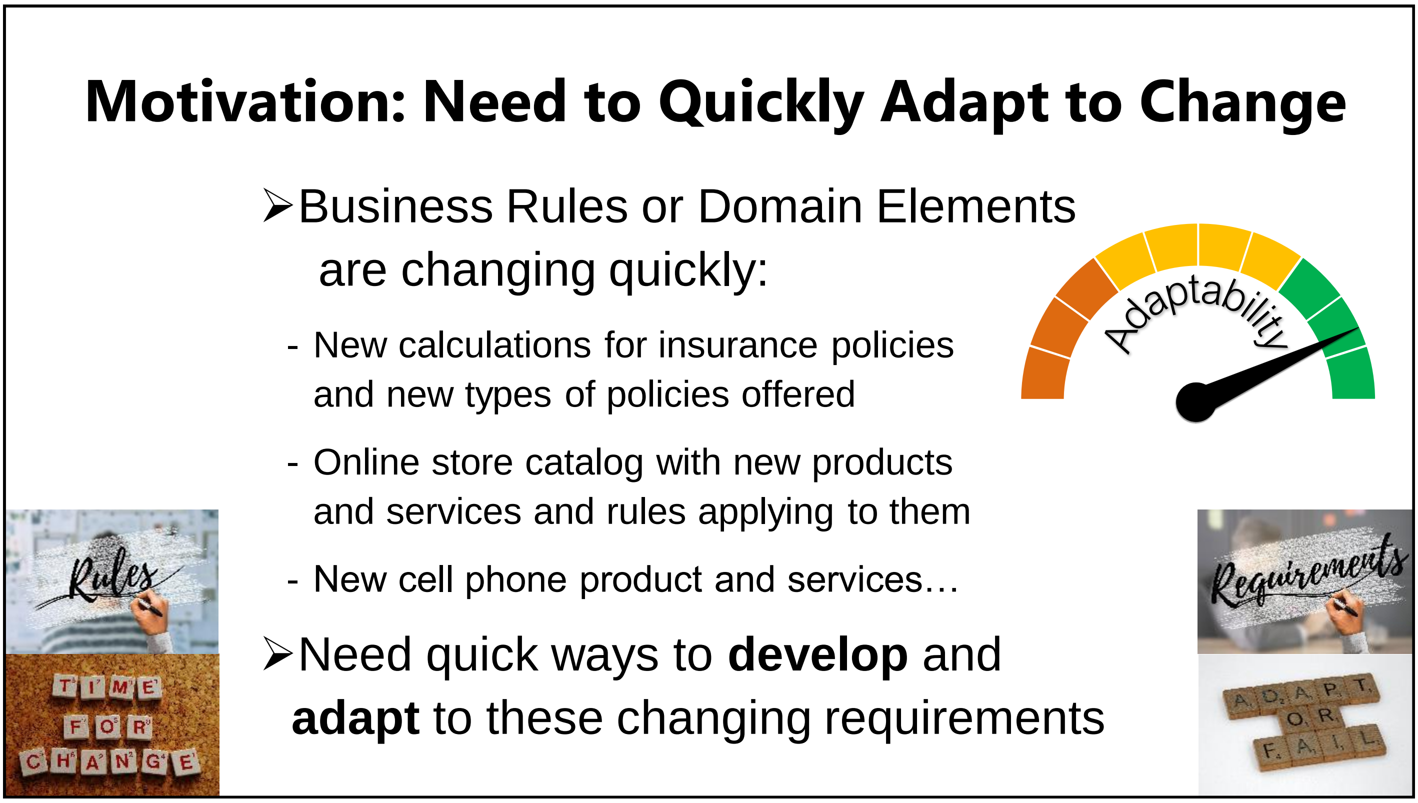


When should I consider Meta-Architectures?

Adaptive Object Models & Meta Modeling

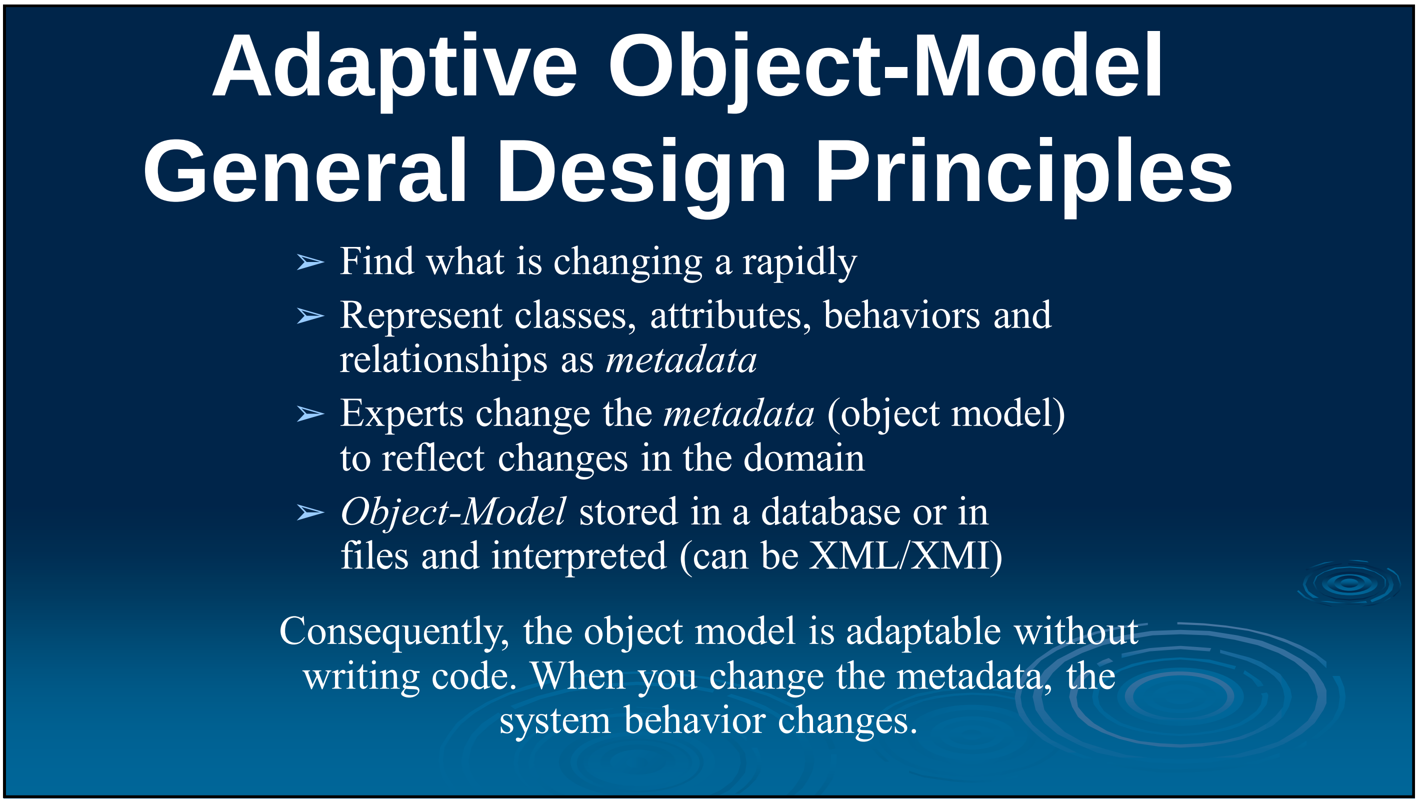


Motivation: Need to Quickly Adapt to Change

➢Business Rules or Domain Elements are changing quickly:

- New calculations for insurance policies and new types of policies offered
- Online store catalog with new products and services and rules applying to them
- New cell phone product and services…

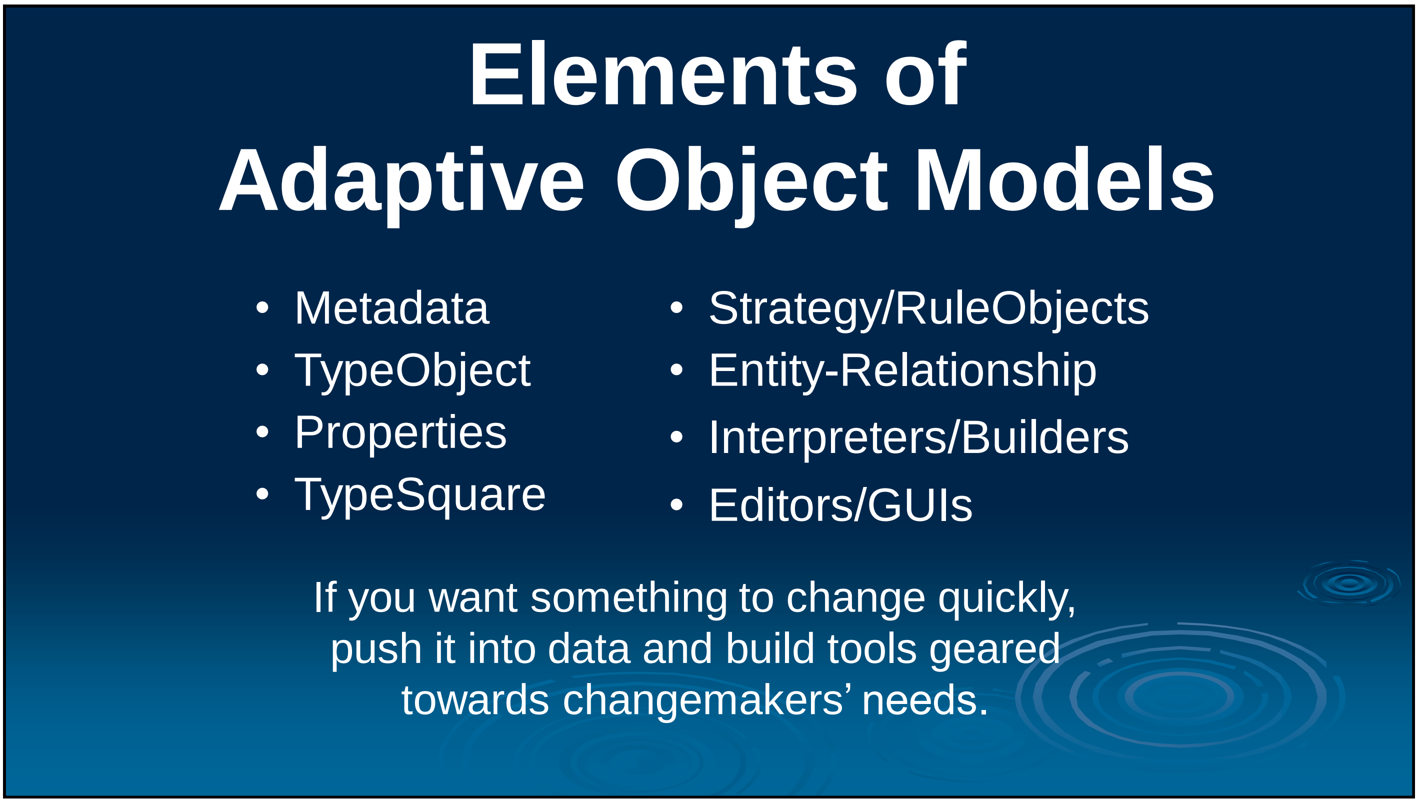
➢Need quick ways to **develop** and **adapt** to these changing requirements



Adaptive Object-Model General Design Principles

- ➢ Find what is changing a rapidly
- ➢ Represent classes, attributes, behaviors and relationships as *metadata*
- ➢ Experts change the *metadata* (object model) to reflect changes in the domain
- ➢ *Object-Model* stored in a database or in files and interpreted (can be XML/XMI)

Consequently, the object model is adaptable without writing code. When you change the metadata, the system behavior changes.



Elements of Adaptive Object Models

- Metadata
- TypeObject
- Properties
- TypeSquare
- Strategy/RuleObjects
- Entity-Relationship
- Interpreters/Builders
- Editors/GUIs

If you want something to change quickly, push it into data and build tools geared towards changemakers' needs.

Drive Practices

Patterns!

What is a Pattern?

Patterns can be thought of “**Good Practices**”

Proven Solutions to **Repeating Problems**

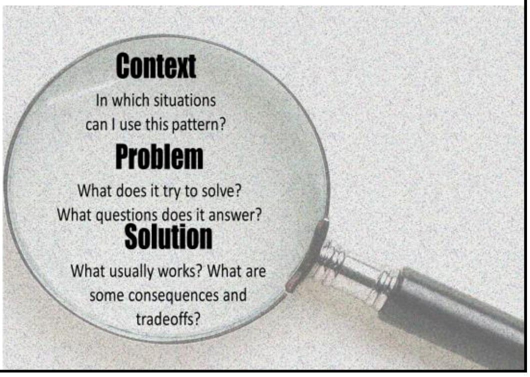
Proven Practices to **Repeating Situations**

Embody Experiences of What Works…
…and What Doesn’t Work

Captures or Describes Knowledge of Experts

Embody “Quality” Attributes for
Solutions to specific Designs

Go to **hillside.net** for more info



Context

In which situations can I use this pattern?

Problem

What does it try to solve? What questions does it answer?

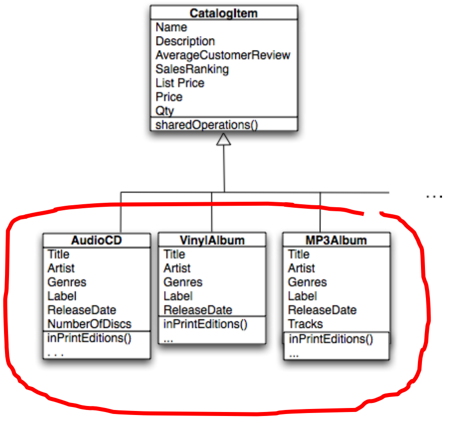
Solution

What usually works? What are some consequences and tradeoffs?

Type-Object

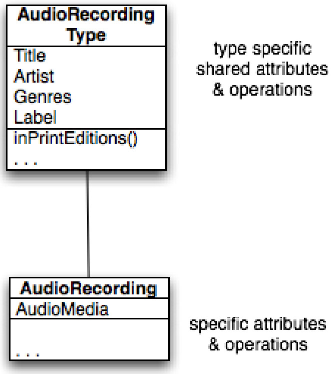
PLoPD3 - Johnson and Woolf

Before



Symptom: Explosion of classes based on minor attribute differences

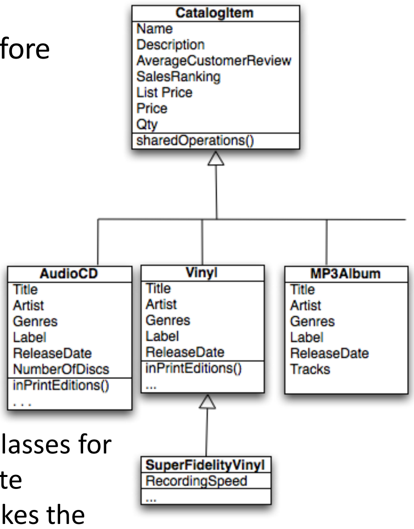
After



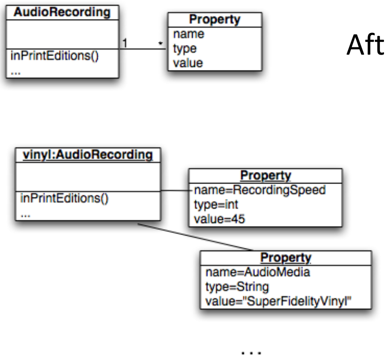
Solution: Factor common attributes into “type” classes

Properties

Before



Creating subclasses for minor attribute variations makes the system static and brittle.



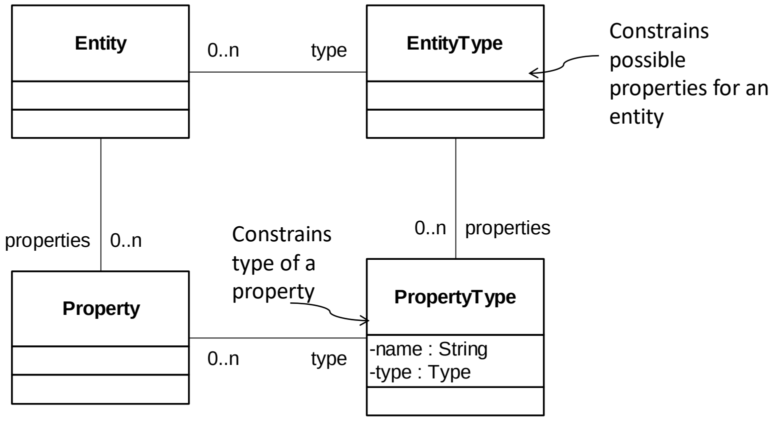
After

Allow instances of a given class to have different attributes. Factor each attribute into a separate Property associated with the class.

But this still isn't flexible enough

- Each time a property is added or changed on its type, the code will need changing.
- How do we define new types of properties?
- How do we validate the proper types?

TypeSquare

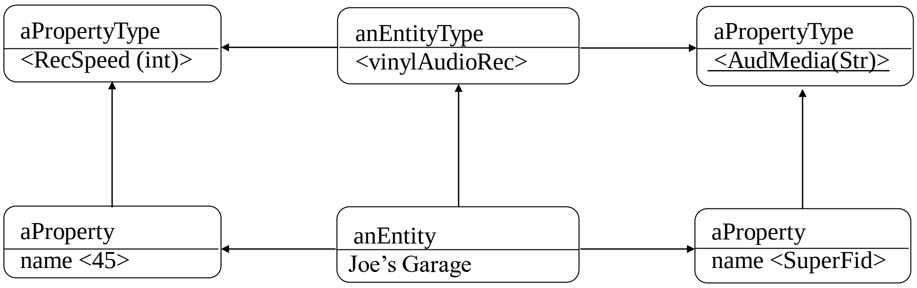


Example: Now it is easy to add different kinds of catalog items

- Sweaters (size=(S,M,L,XL), color=(red,green,blue,yellow,…)
- Canoes (length=float, width=float)

Type Square

(instance diagram)



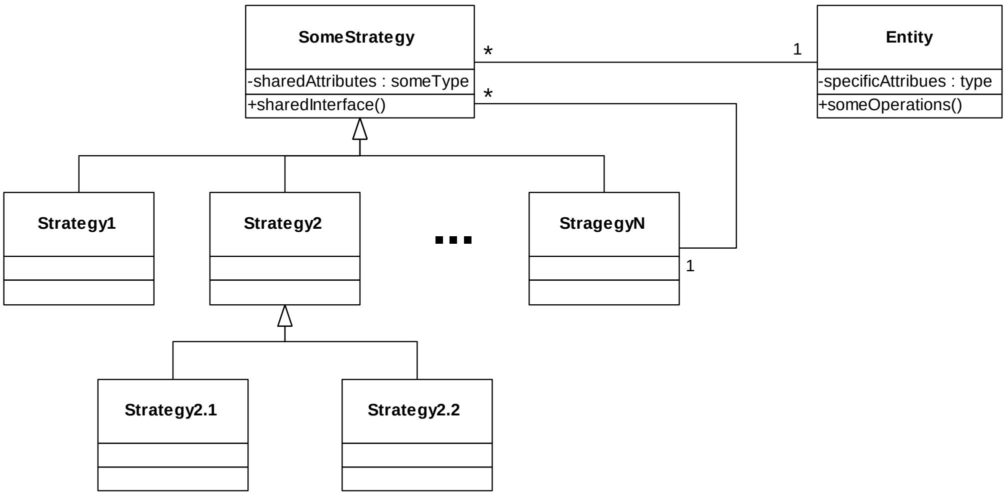
Dealing with Behavior/Rules

- ➢ Making methods that implement the different algorithm for each Type or Property could require a large case-statement and could be impractical to maintain.
- ➢ Instances for the similar types can have different algorithm depending upon context.

The model has to implements a defined set of interchangeable algorithms that customize the behavior of the system.

Strategies/RuleObjects Solution

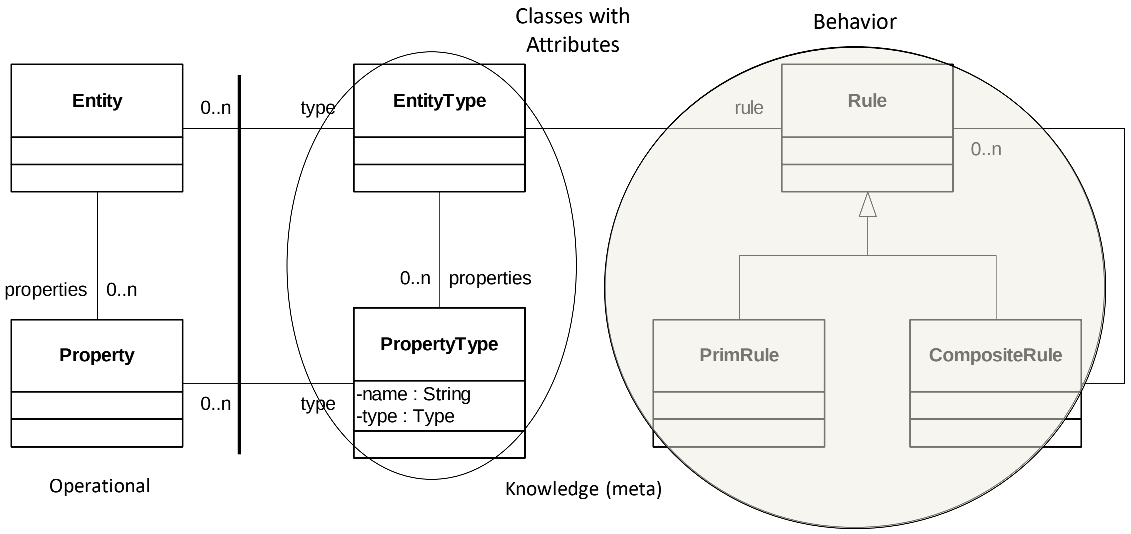
(Behavior/Methods)



Design Patterns - GOF95

Putting It All Together

(Very Common Structure)



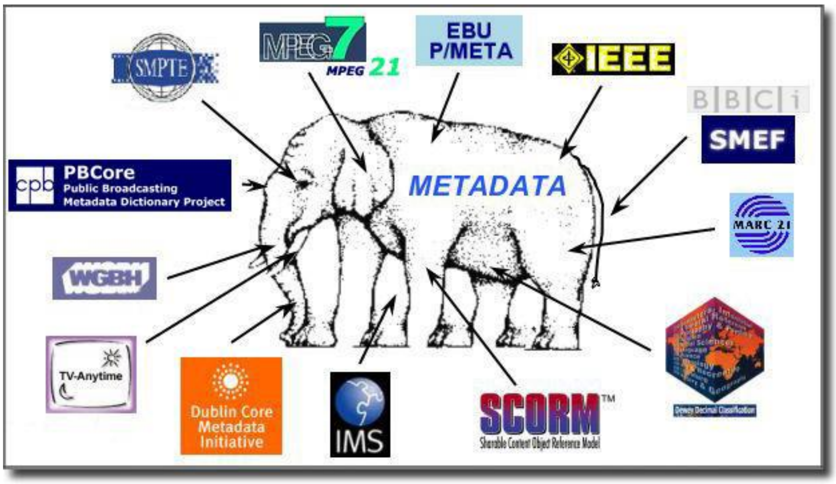
ECOOP & OOPSLA 2001 Yoder, Balaguer, Johnson

In my youth…two bad words

M and R words

"Metadata and Reflection"

Metadata


SMPTE
MPEG 7
MPEG 21
EBU P/META
IEEE
BBCi
SMEF
PBCore
cpb
Public Broadcasting Metadata Dictionary Project
METADATA
MARC 21
WGBH
TV-Anytime
Dublin Core Metadata Initiative
IMS
SCORM™
Sharable Content Object Reference Model
Dewey Decimal Classification

The Power of Metadata

Code is data, data is code. Everything is data. Data drives the behavior.

"Anything you can do, I can do Meta"

Meta data simply describes other data.

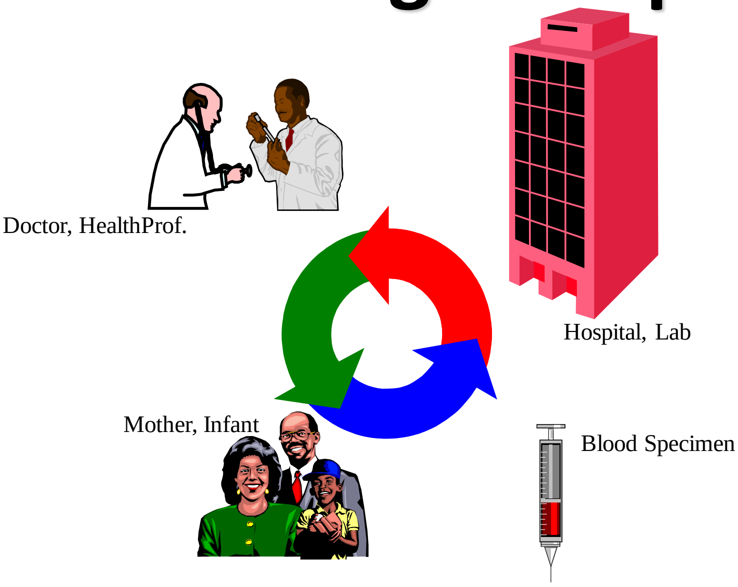
"If something is going to vary in a predictable way, store the description of the variation in a database so that it is easy to change" – Ralph Johnson

"Meta is Beta"

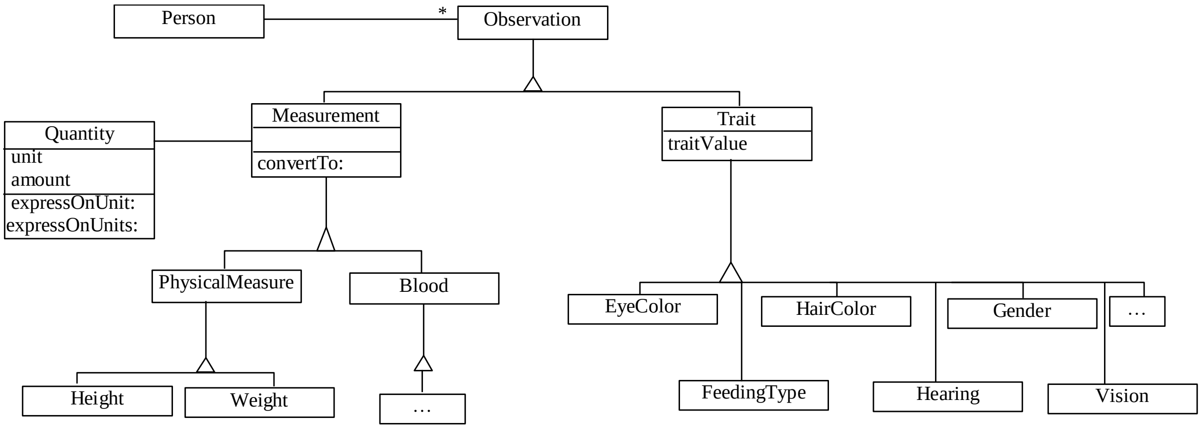
An AOM Example…

Refactoring as We Go

Newborn Screening Refactoring Example



Medical Observation – Basic OO Design Model



Person

*

Observation

Measurement

convertTo:

Quantity

unit
amount

expressOnUnit:
expressOnUnits:

Trait

traitValue

PhysicalMeasure

Blood

Height

Weight

…

EyeColor

HairColor

Gender

…

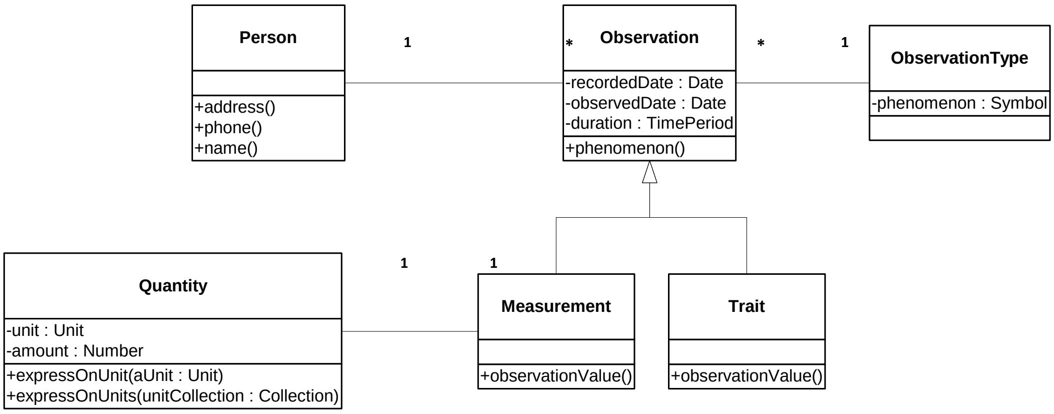
FeedingType

Hearing

Vision

What happens when a new observation is required?

Observation Design (1st Design)



Person

+address()
+phone()
+name()

1

*

Observation

-recordedDate : Date
-observedDate : Date
-duration : TimePeriod

+phenomenon()

*

1

ObservationType

-phenomenon : Symbol

Quantity

-unit : Unit
-amount : Number

+expressOnUnit(aUnit : Unit)
+expressOnUnits(unitCollection : Collection)

1

1

Measurement

+observationValue()

Trait

+observationValue()

Observation Design

Example

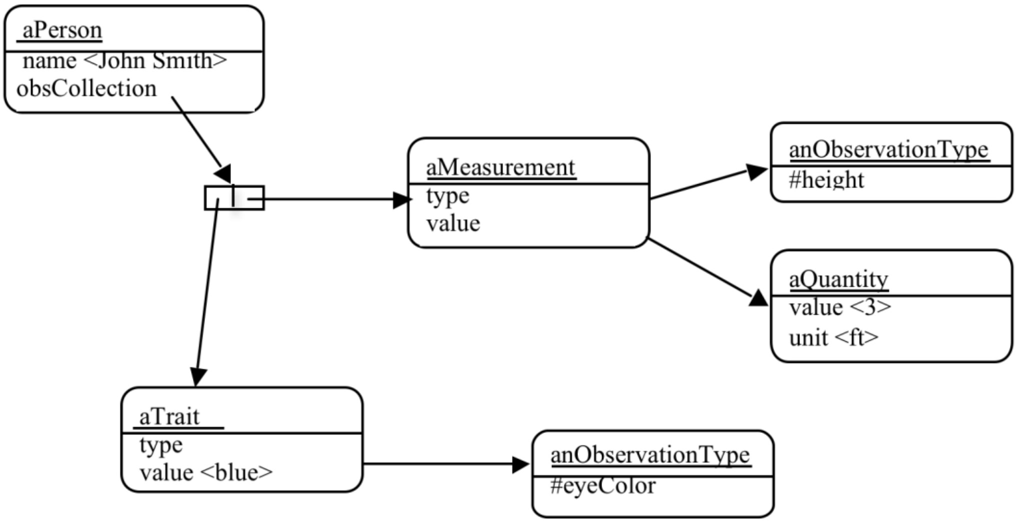
Name: **John Smith**
Mother: Sue Smith
Father:
Address:
Phone:

Height: **3 feet**

Eyes Color: **Blue**

Observation Design

(instance diagram)



aPerson
name <John Smith>
obsCollection

aMeasurement
type
value

anObservationType
#height

aQuantity
value <3>
unit <ft>

aTrait
type
value <blue>

anObservationType
#eyeColor

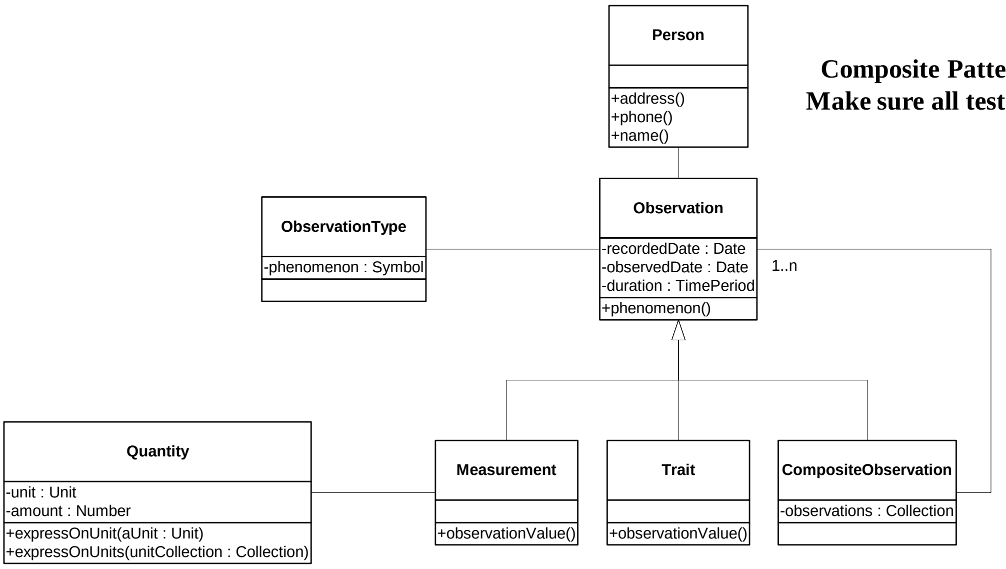
Composing Observations

Observations can be more complex

- Cholesterol
  - Components: HDL, LDL
- Blood Pressure
  - Components: Systolic, Diastolic
- Vision
  - Components: Left Eye, Right Eye

Composite Observation Design

(1st Refactoring)



Person
+address()
+phone()
+name()

Observation
-recordedDate : Date
-observedDate : Date
-duration : TimePeriod
+phenomenon()

ObservationType
-phenomenon : Symbol

1..n

Quantity
-unit : Unit
-amount : Number
+expressOnUnit(aUnit : Unit)
+expressOnUnits(unitCollection : Collection)

Measurement
+observationValue()

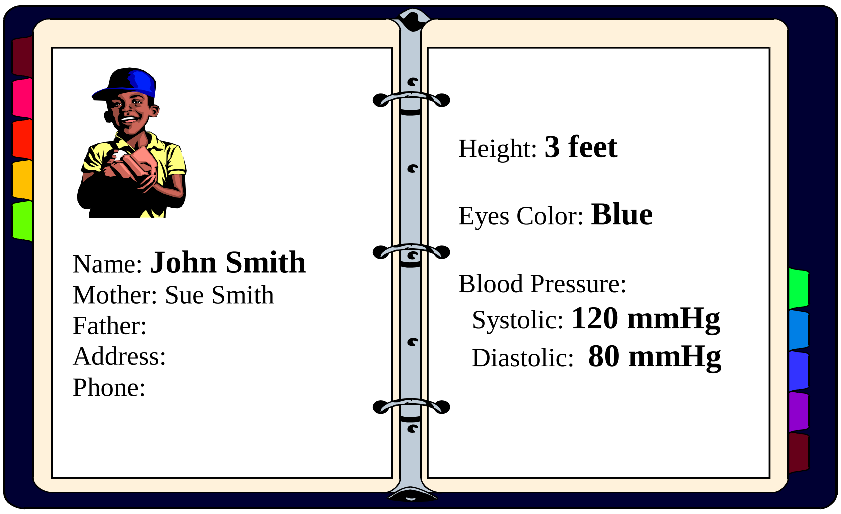
Trait
+observationValue()

CompositeObservation
-observations : Collection

Composite Pattern (GOF)
Make sure all tests still pass!

Observation Design

Example



Name: **John Smith**
Mother: Sue Smith
Father:
Address:
Phone:

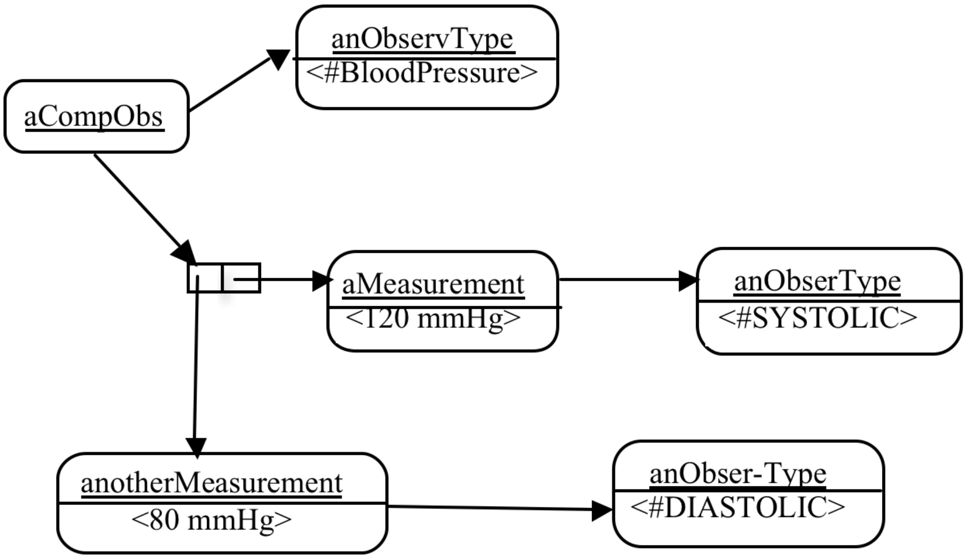
Height: **3 feet**

Eyes Color: **Blue**

Blood Pressure:
Systolic: **120 mmHg**
Diastolic: **80 mmHg**

Composite Observation Design

(instance diagram)



aCompObs

anObservType
<#BloodPressure>

aMeasurement
<120 mmHg>

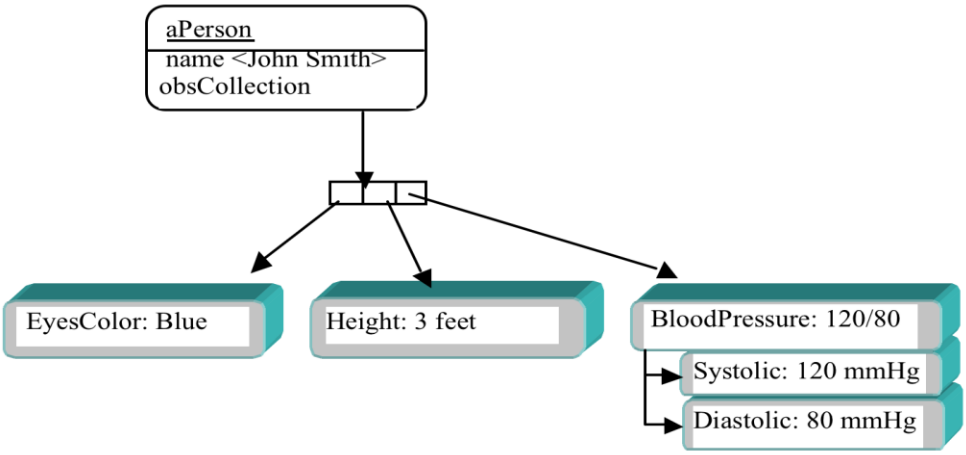
anObserType
<#SYSTOLIC>

anotherMeasurement
<80 mmHg>

anObser-Type
<#DIASTOLIC>

Composite and Primitive Observation Design

What we know about John?

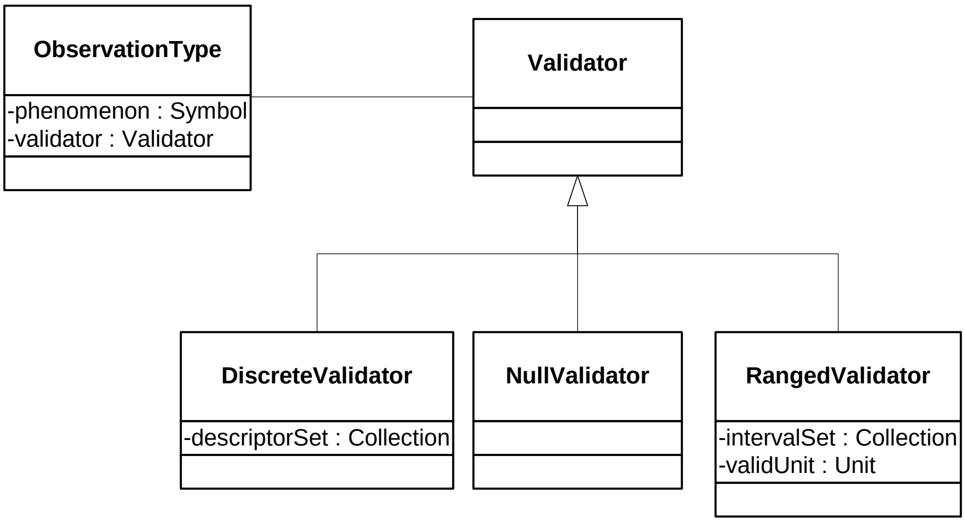


Validating Observations

- Each Observation has its own set of legal values:
  - Baby's Weight: [0..30] pounds
  - HepatitisB: {positive, negative}
  - Left/Right Vision: {normal, abnormal}
- The GUI could enforce legal values
  - but we prefer these business rules in domain objects

Validating Observations Design

(2nd Refactoring)



ObservationType

-phenomenon : Symbol
-validator : Validator

Validator

DiscreteValidator

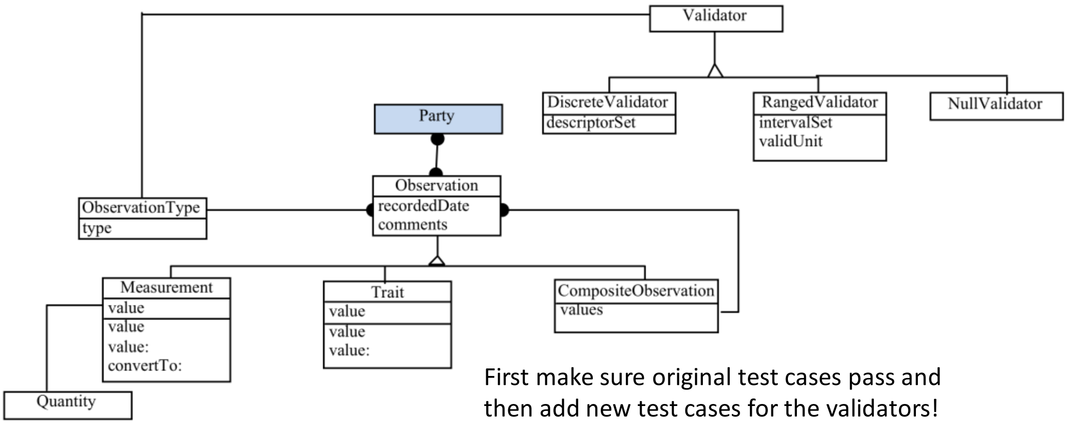
-descriptorSet : Collection

NullValidator

RangedValidator

-intervalSet : Collection
-validUnit : Unit

Overall Observation Design



Validator

DiscreteValidator
descriptorSet

RangedValidator
intervalSet
validUnit

NullValidator

Party

Observation
recordedDate
comments

ObservationType
type

Measurement
value
value
value:
convertTo:

Quantity

Trait
value
value
value:

CompositeObservation
values

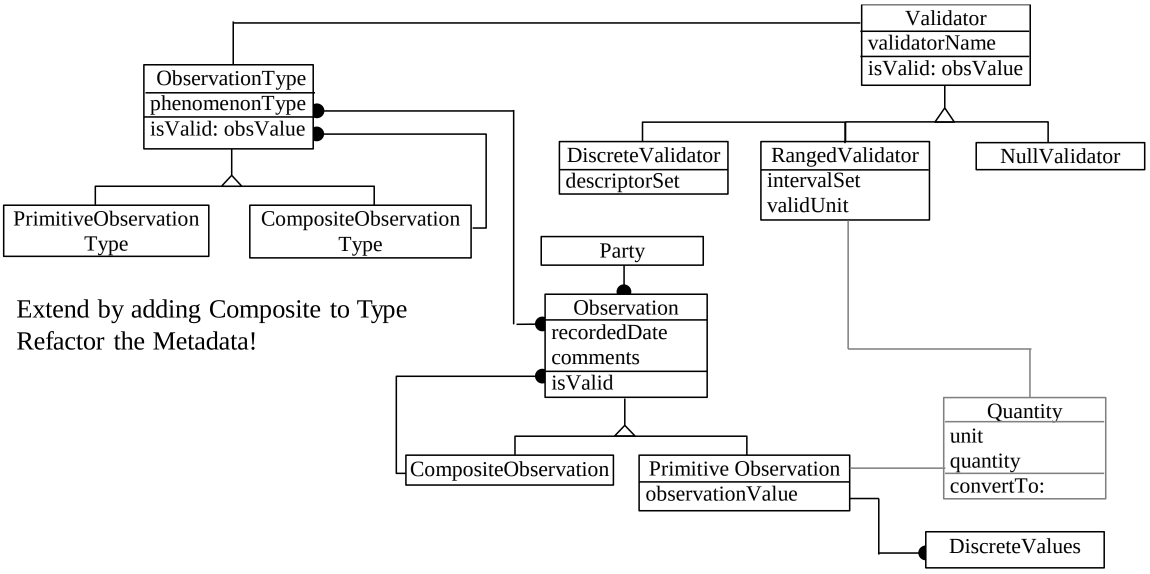
First make sure original test cases pass and then add new test cases for the validators!

Is everything an Observation?

How does the model specify the structure of the Composite?

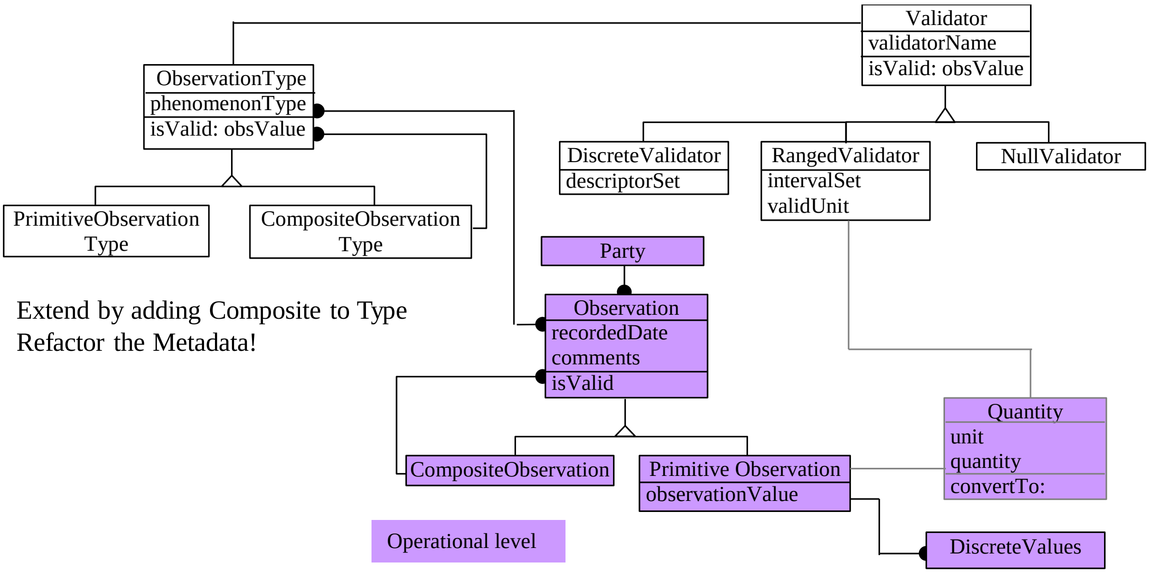
What is the relationship between Trait and DiscreteValidator?

Observation Design



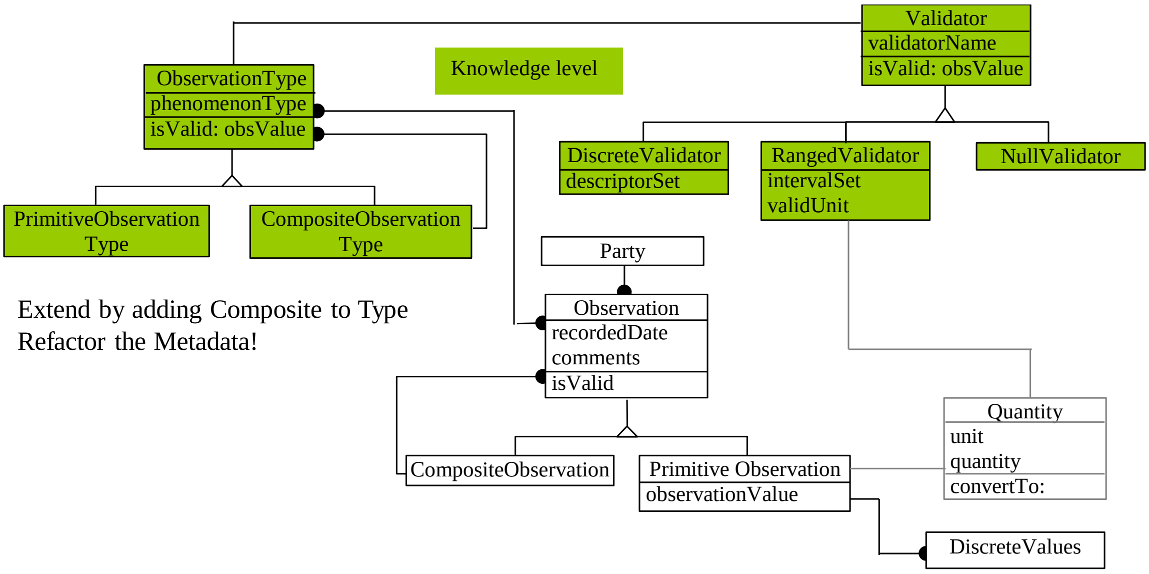
Extend by adding Composite to Type
Refactor the Metadata!

Observation Design



Extend by adding Composite to Type
Refactor the Metadata!

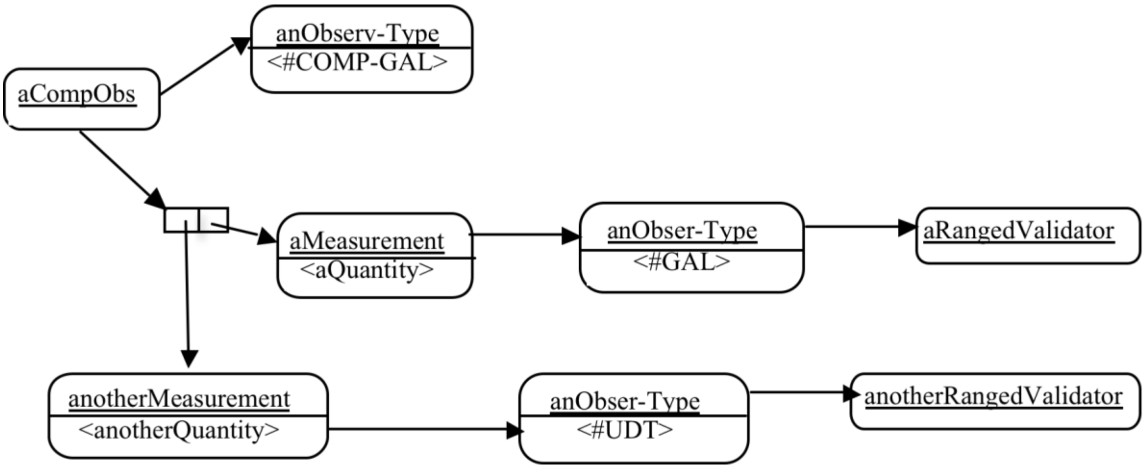
Observation Design



Extend by adding Composite to Type
Refactor the Metadata!

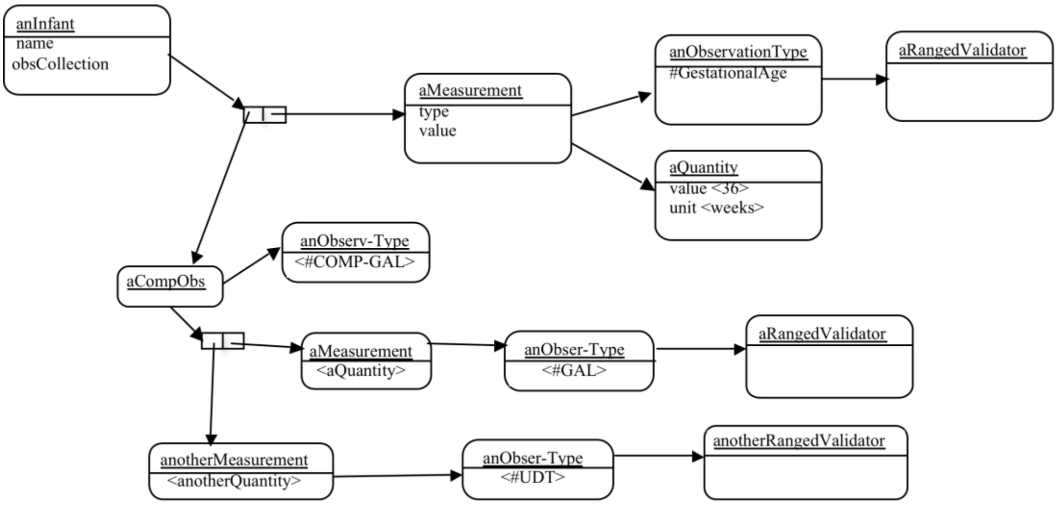
Observation Design

(instance diagram)

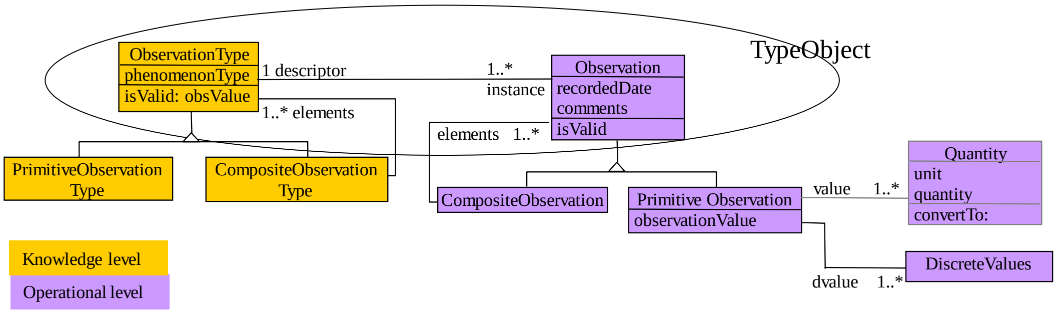


Observation Design

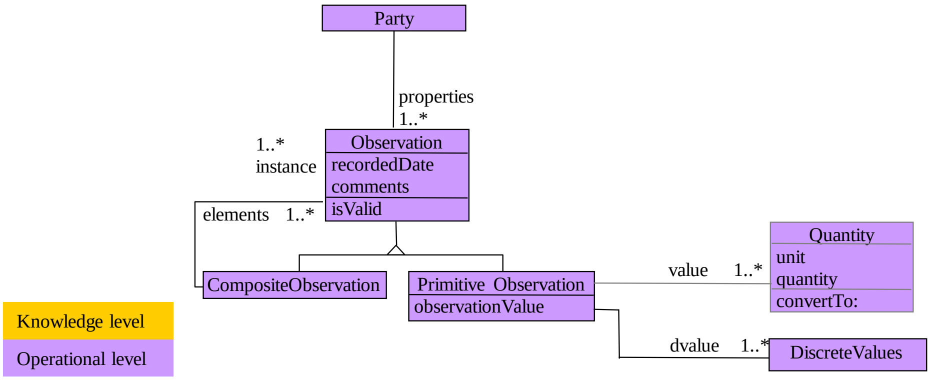
(instance diagram)



Observations: TypeObject



Observations: Properties



Party

properties 1..*

1..* instance

Observation
recordedDate
comments
isValid

elements 1..*

CompositeObservation

Primitive Observation
observationValue

value 1..*

Quantity
unit
quantity
convertTo:

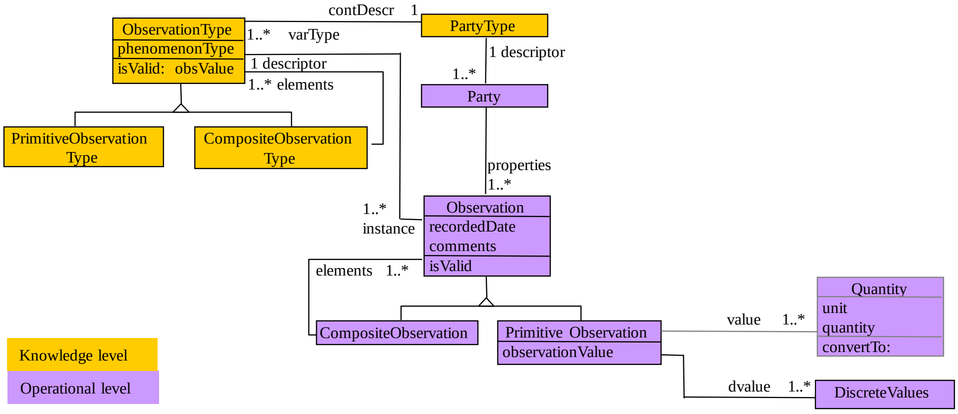
dvalue 1..*

DiscreteValues

Knowledge level

Operational level

Observations: TypeSquare



ObservationType
phenomenonType
isValid: obsValue

contDescr 1

1..* varType

PartyType

1 descriptor

1..*

1 descriptor

1..* elements

Party

PrimitiveObservation Type

CompositeObservation Type

properties 1..*

1..* instance

Observation
recordedDate
comments
isValid

elements 1..*

CompositeObservation

Primitive Observation
observationValue

value 1..*

Quantity
unit
quantity
convertTo:

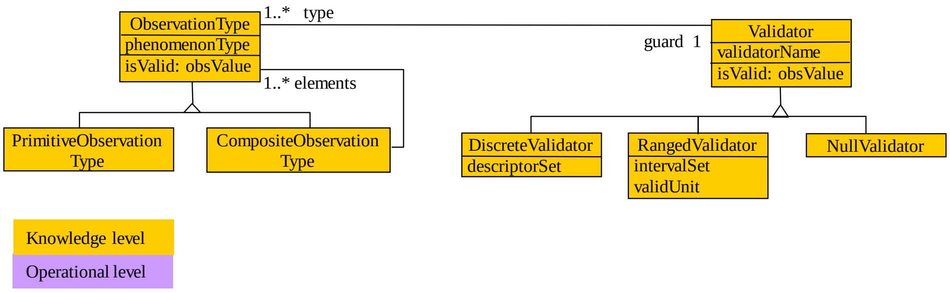
dvalue 1..*

DiscreteValues

Knowledge level

Operational level

Observations: Strategy



ObservationType
phenomenonType
isValid: obsValue

1..* type

guard 1

Validator
validatorName
isValid: obsValue

1..* elements

PrimitiveObservation Type

CompositeObservation Type

DiscreteValidator
descriptorSet

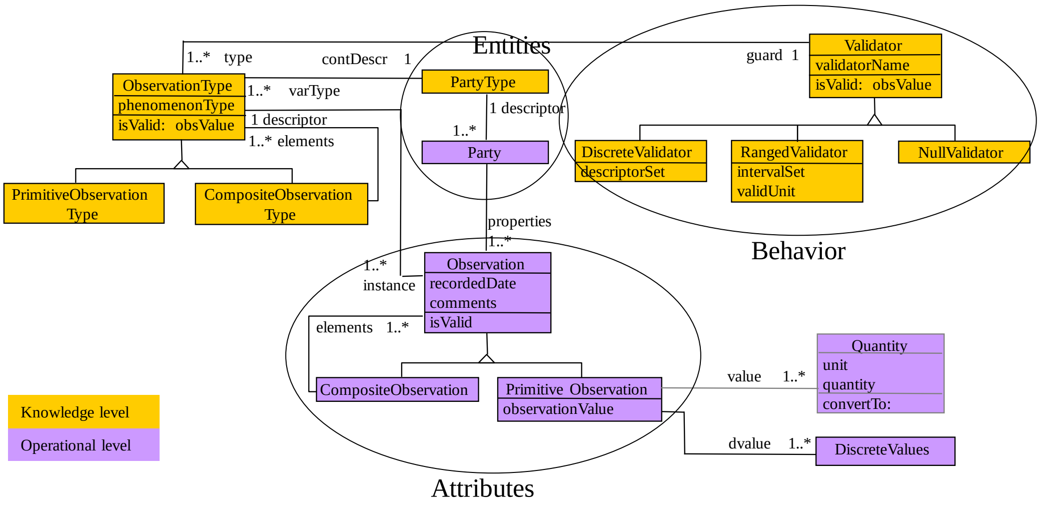
RangedValidator
intervalSet
validUnit

NullValidator

Knowledge level

Operational level

Medical Observations Design



1..* type

contDescr 1

Entities

guard 1

Validator
validatorName
isValid: obsValue

ObservationType
phenomenonType
isValid: obsValue

1..* varType

PartyType

1 descriptor

1 descriptor

1..* elements

1..*

Party

DiscreteValidator
descriptorSet

RangedValidator
intervalSet
validUnit

NullValidator

PrimitiveObservation Type

CompositeObservation Type

properties
1..*

Behavior

1..*
instance

Observation
recordedDate
comments
isValid

elements 1..*

CompositeObservation

Primitive Observation
observationValue

value 1..*

Quantity
unit
quantity
convertTo:

dvalue 1..*

DiscreteValues

Attributes

Knowledge level

Operational level

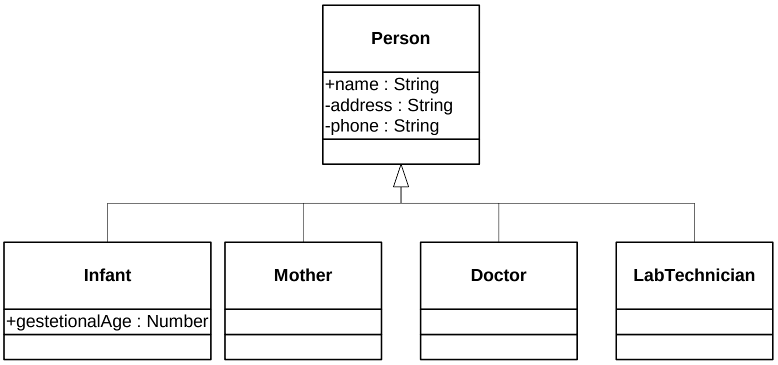
Refactoring Leverage

- Refactoring exploits Brooks' "promising attacks" from *No Silver Bullet*:
  - grow don't build software: software growth involves restructuring (this is core to Agile);
  - requirements refinements and rapid prototyping: refactoring supports such design exploration, and adapting to changing customer needs;
  - support great designers: refactoring is yet another tool in a designer's tool chest.

Extending our Example to Include…

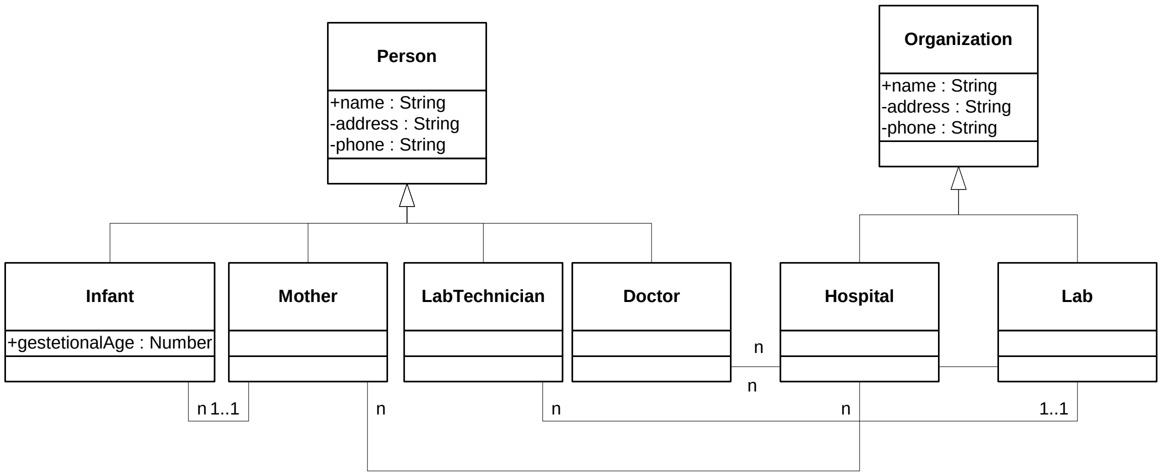
Entities and Relationships

Infants, Mothers and Doctors...

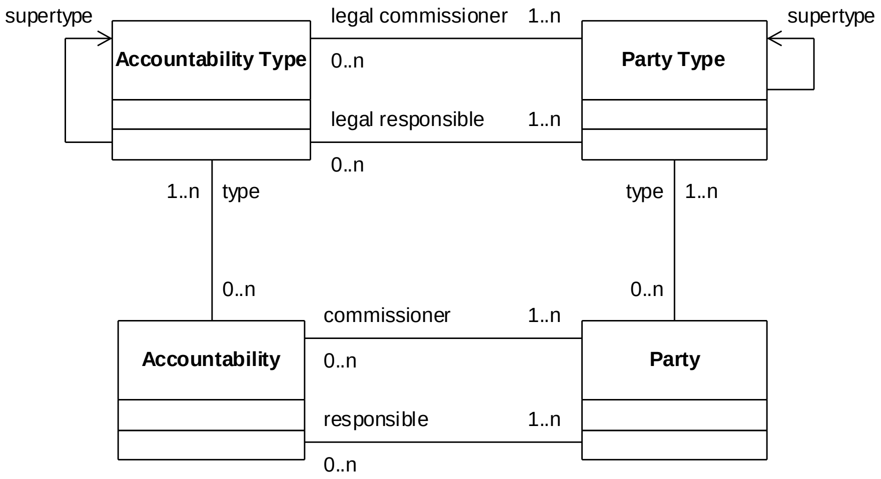


Newborn Screening

Putting it all together



Entity-Relationship Patterns



supertype

Accountability Type

legal commissioner 1..n

0..n

Party Type

supertype

legal responsible 1..n

0..n

1..n type

type 1..n

0..n

0..n

commissioner 1..n

Accountability

0..n

Party

responsible 1..n

0..n

Analysis Patterns – Martin Fowler

Party and Accountability

Modeling relationships between entities

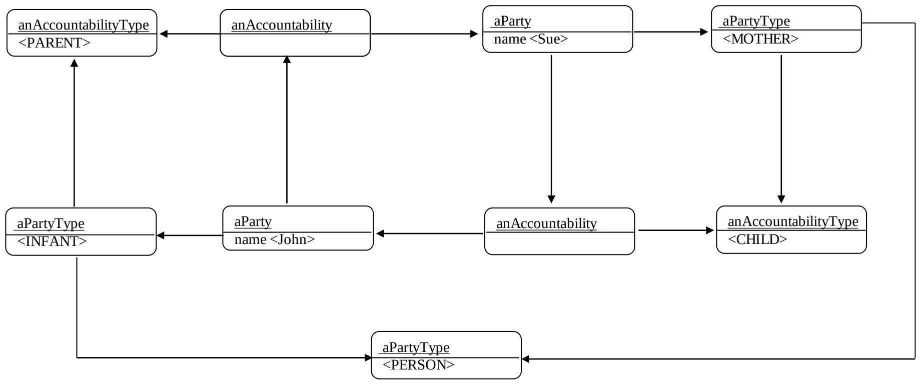
Sue Smith

John Smith

Sue is the mother of John

Party and Accountability

(instance diagram)



anAccountabilityType
<PARENT>

anAccountability

aParty
name <Sue>

aPartyType
<MOTHER>

aPartyType
<INFANT>

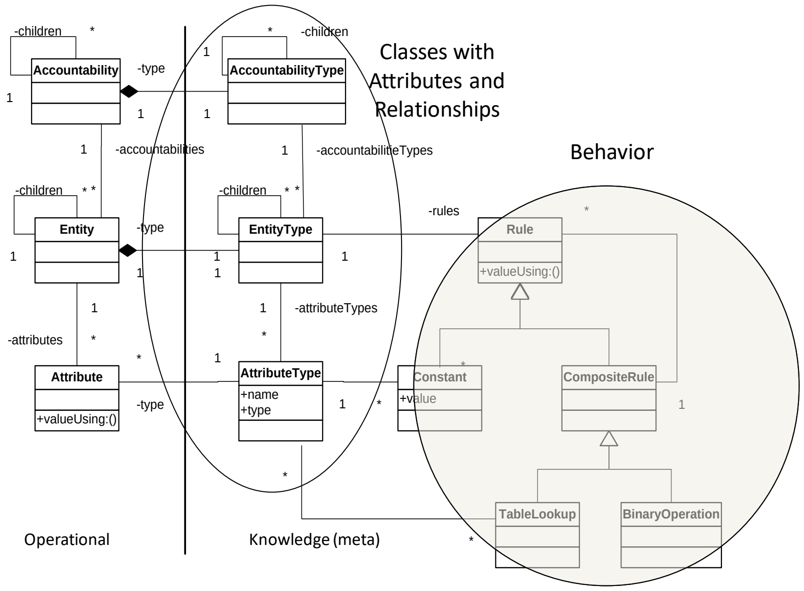
aParty
name <John>

anAccountability

anAccountabilityType
<CHILD>

aPartyType
<PERSON>

Putting it All Together: Adaptive Object Model "Core Architecture"



Classes with Attributes and Relationships

Behavior

Accountability

AccountabilityType

Entity

EntityType

Attribute
+valueUsing:()

AttributeType
+name
+type

Rule
+valueUsing:()

Constant
+value

CompositeRule

TableLookup

BinaryOperation

-children

-type

-accountabilities

-accountabilitieTypes

-attributes

-attributeTypes

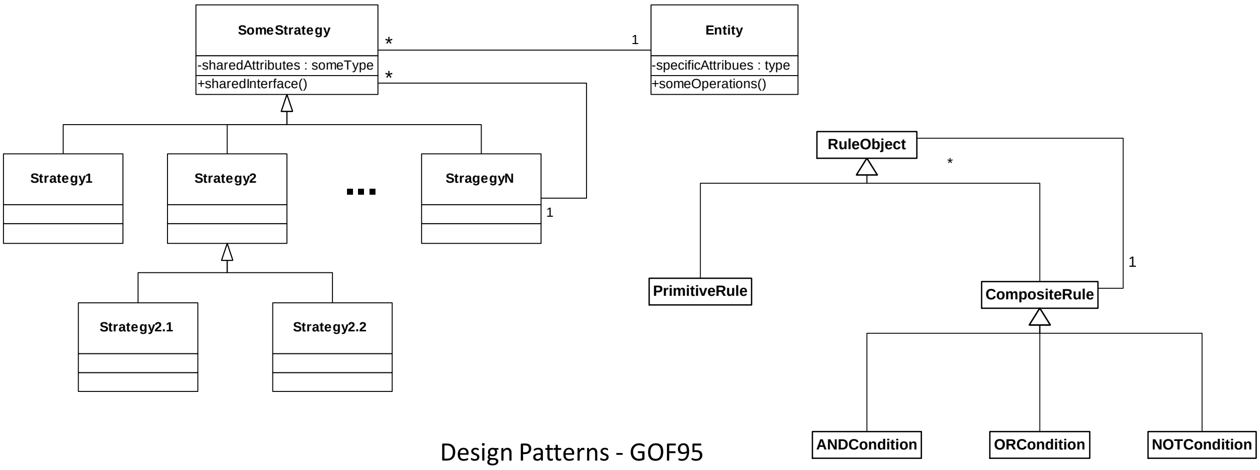
-rules

Operational

Knowledge (meta)

Strategies/Interpreters/RuleObjects

(Behavior/Methods)



Design Patterns - GOF95

Composite Strategies ➔ Interpreter

Composite Strategies

Problem: Strategy leads to a big class hierarchy, one class for each kind of policy

Solution: Make Composite Strategies using Primitive Operations

=> Interpreter pattern

What About Roles?

Problem: How do you deal with dynamic behavior for an object? For example, a person can be either a mother, child, or doctor in our system.

Solution: Create a Role Object that defines their behavior. A “role” defines a pluggable strategy.

Roles

(Parties, Accountabilities and Properties
is the Beginning of Roles)

- Babies
  - Have Mothers and Doctors
  - Gestational Age,
  - Hearing and Vision,
  - Weight, Race, Ethnicity, DOB, ...
- Mothers
  - Have Babies and Doctors
  - Hepatitus present at Birth (y/n),
  - Languages, Race, Ethnicity, ...

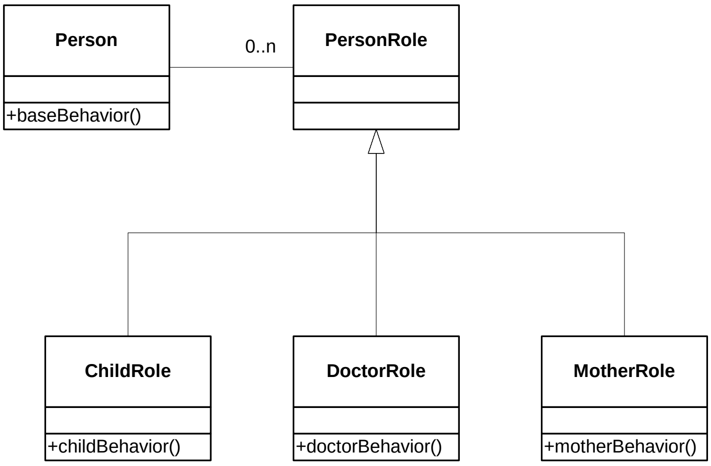
Roles

(Parties, Accountabilities and Properties
is the Beginning of Roles)

- In our system, there are different types of parties, relationships between them, and properties on the parties, including different observations.
- The pluggable behavior (or different roles) is defined for a given party by the legal relationships it can have and the set of properties that are allowed.

Roles

(an example)

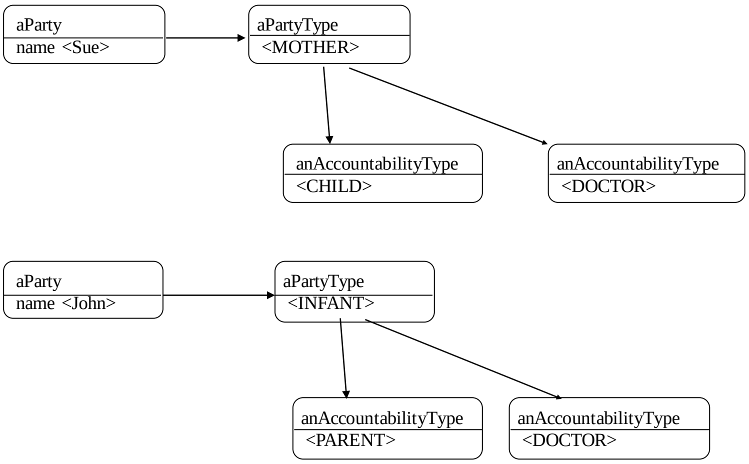


PLoP 97 - Fowler

PLoPD4 - Baumer, Riehle, Siberski, Wulf

Roles

(Parties, Accountabilities and Properties is the Beginning of Roles)



We have examined the “core” patterns for the domain model

What else is there?

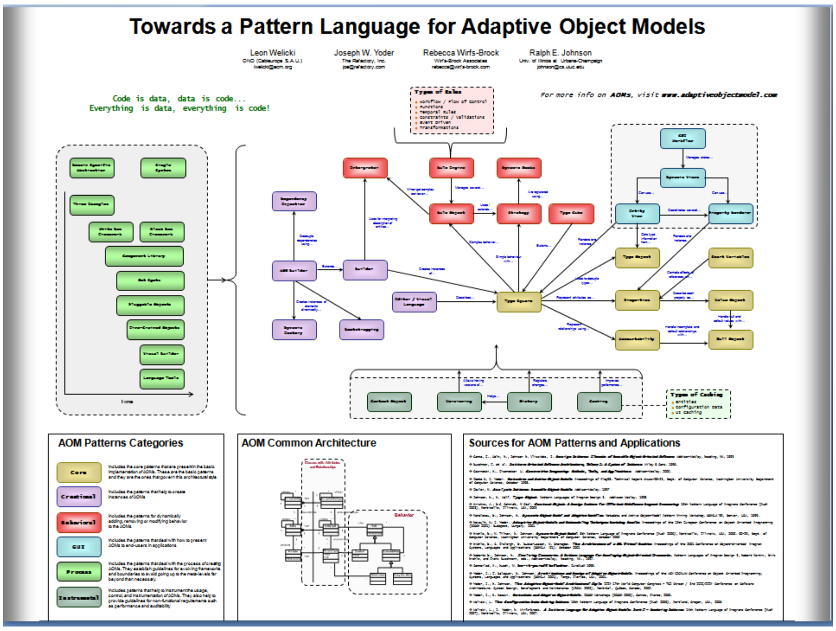
How do you interact with the domain?

…

We Have Only Shown Part of a Larger AOM Pattern Language

- **Core Patterns**: the basic implementation of AOM domain objects.
- **Presentation Patterns**: how to visually represent AOMs.
- **Creational**: how to create instances of domain objects.
- **Behavioral**: dynamically adding, removing or modifying behavior (business rules).
- **Process Patterns**: the process of creating AOMs. They establish guidelines for evolving frameworks and boundaries to avoid implementing meta beyond what's necessary.
- **Miscellaneous**: usage, control, and instrumentation of AOMs and guidelines for non-functional requirements such as performance or auditability.

OOPSLA Poster Session



Towards a Pattern Language for Adaptive Object Models

Leon Welicki
OND (Cableuropa S.A.U.)
lwelicki@acm.org

Joseph W. Yoder
The Refactory, Inc.
joe@refactory.com

Rebecca Wirfs-Brock
Wirfs-Brock Associates
rebecca@wirfs-brock.com

Ralph E. Johnson
Univ. of Illinois at Urbana-Champaign
johnson@cs.uiuc.edu

Code is data, data is code...
Everything is data, everything is code!

For more info on AOMs, visit www.adaptiveobjectmodel.com

Types of Rules
- workflow / flow of control
- functions
- temporal rules
- constraints / validations
- event driven
- transformations

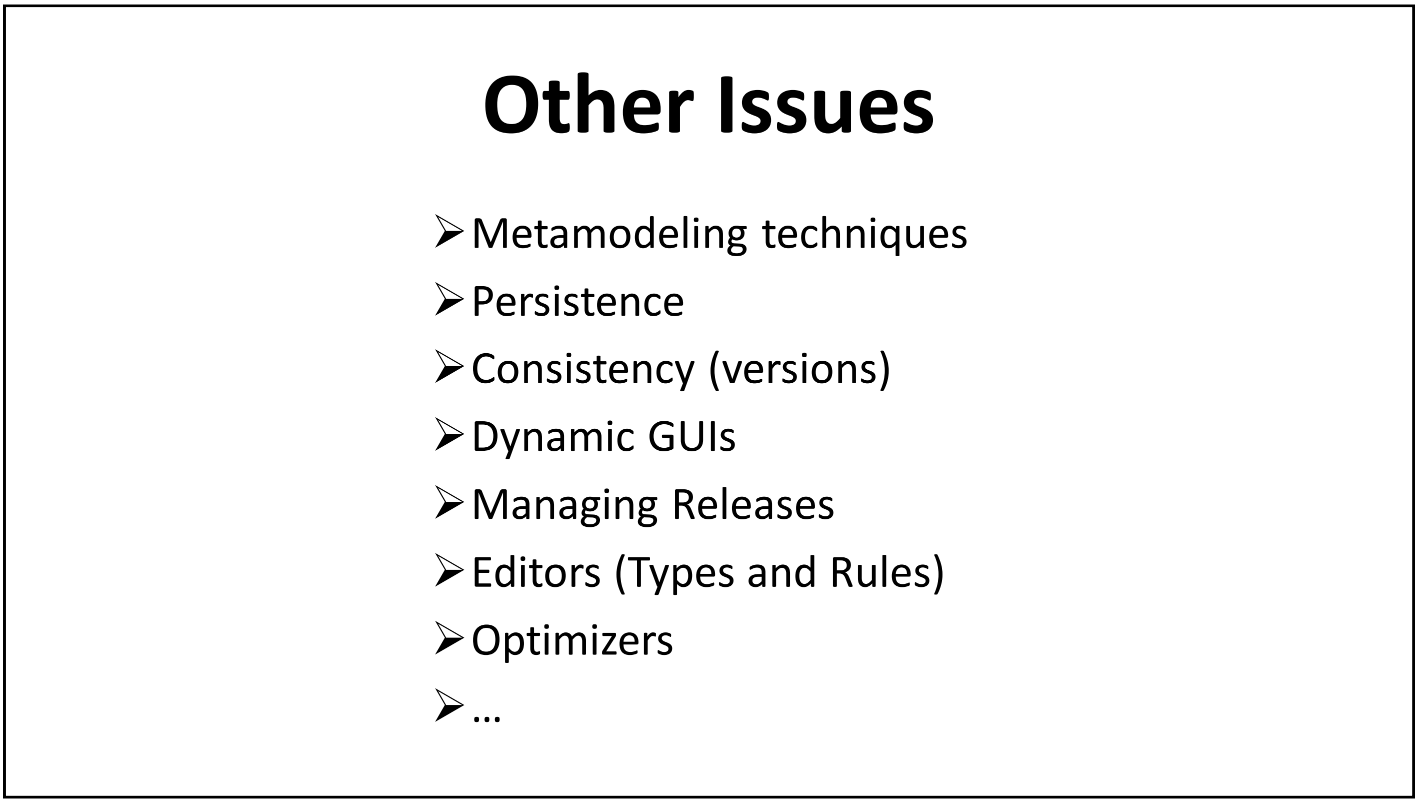
Types of Caching
- entities
- configuration data
- ui caching

AOM Patterns Categories

Core	Includes the core patterns that are present in the basic implementation of AOMs. These are the basic patterns and they are the ones that govern this architectural style
Creational	Includes the patterns that help to create instances of AOMs
Behavioral	Includes the patterns for dynamically adding, removing or modifying behavior to the AOMs
GUI	Includes the patterns that deal with how to present AOMs to end-users in applications
Process	Includes the patterns that deal with the process of creating AOMs. They establish guidelines for evolving frameworks and boundaries to avoid going up to the meta-levels far beyond than necessary
Instrumental	Includes patterns that help to instrument the usage, control, and instrumentation of AOMs. They also help to provide guidelines for non-functional requirements such as performance and auditability

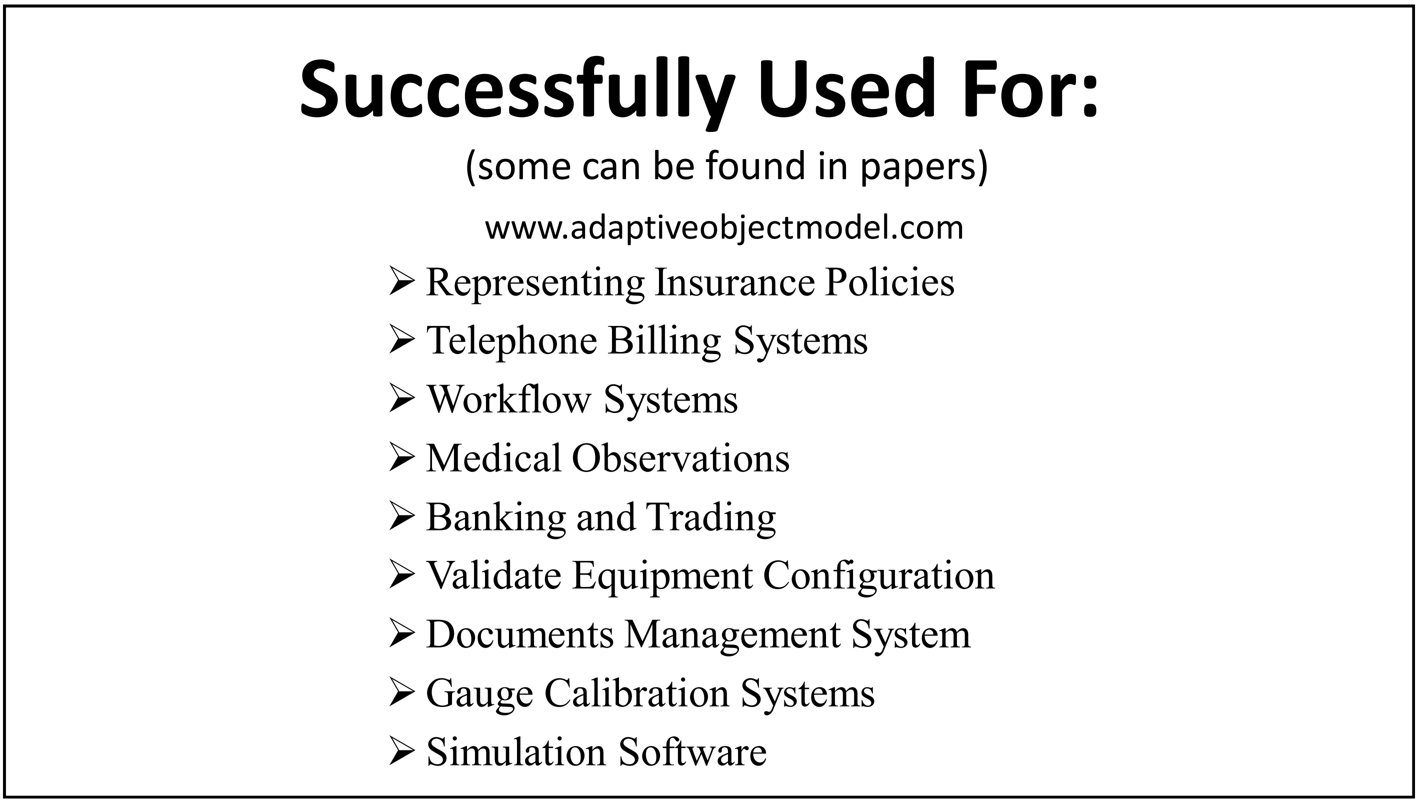
AOM Common Architecture

Sources for AOM Patterns and Applications



Other Issues

- ➢ Metamodeling techniques
- ➢ Persistence
- ➢ Consistency (versions)
- ➢ Dynamic GUIs
- ➢ Managing Releases
- ➢ Editors (Types and Rules)
- ➢ Optimizers
- ➢ …

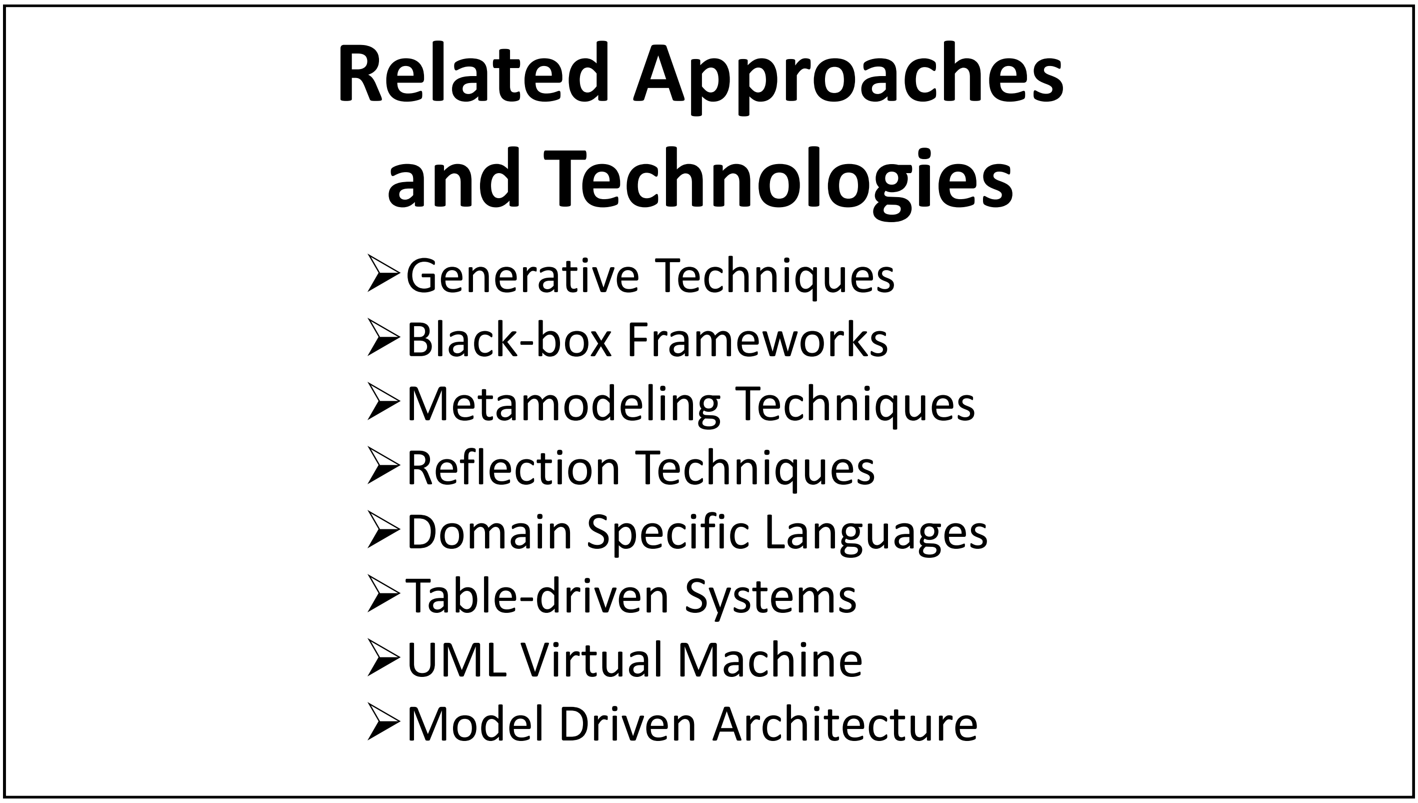


Successfully Used For:

(some can be found in papers)

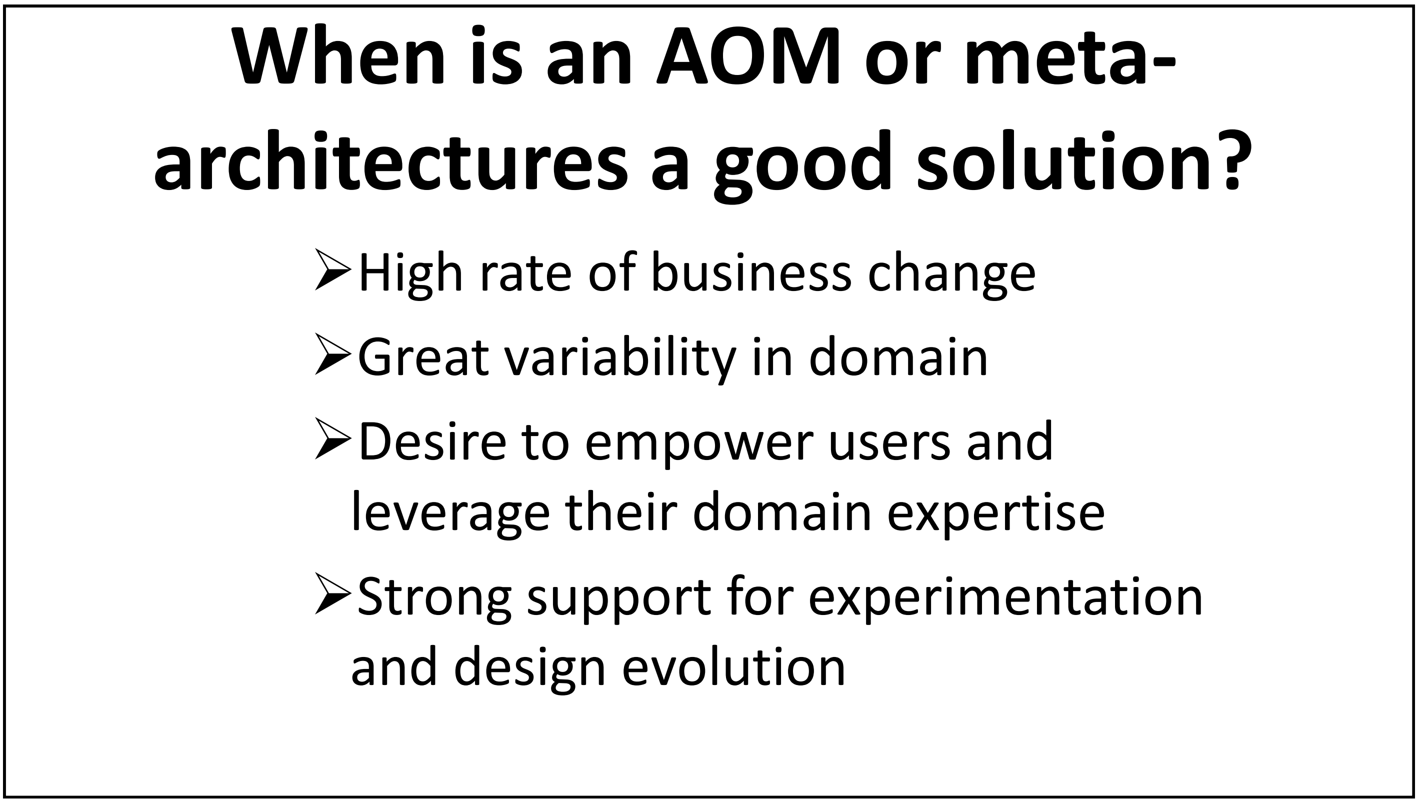
www.adaptiveobjectmodel.com

- ➢ Representing Insurance Policies
- ➢ Telephone Billing Systems
- ➢ Workflow Systems
- ➢ Medical Observations
- ➢ Banking and Trading
- ➢ Validate Equipment Configuration
- ➢ Documents Management System
- ➢ Gauge Calibration Systems
- ➢ Simulation Software



Related Approaches and Technologies

- Generative Techniques
- Black-box Frameworks
- Metamodeling Techniques
- Reflection Techniques
- Domain Specific Languages
- Table-driven Systems
- UML Virtual Machine
- Model Driven Architecture



When is an AOM or meta-architectures a good solution?

- High rate of business change
- Great variability in domain
- Desire to empower users and leverage their domain expertise
- Strong support for experimentation and design evolution

The Business Case for an Adaptive Object-Model System

- Higher overall ROI
- Better domain flexibility
- Fosters business innovation
- Supports business “ownership”
- Can be done incrementally via prototyping and design evolution

Downside of Meta-Architectures

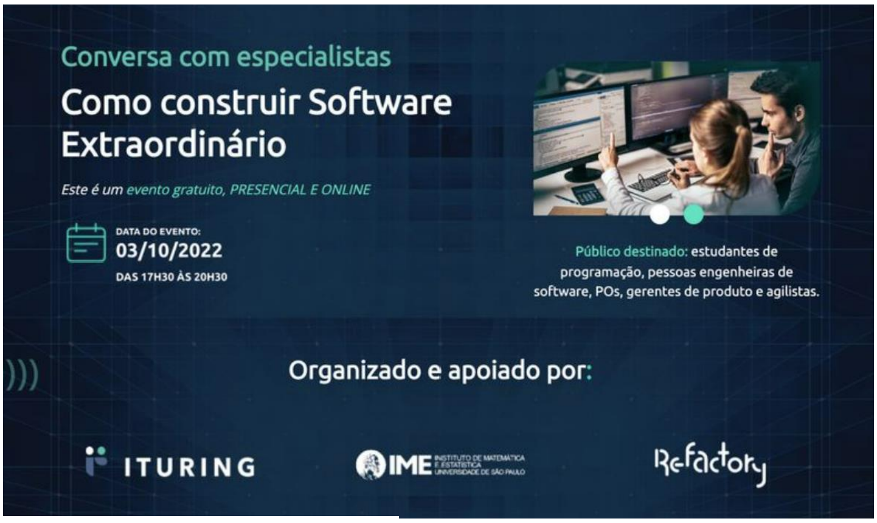
- Requires Skilled People
- Performance / Security
- Lack of Support/Tools
- Misused/Abused
- Complexity / Over Design
- …

Reasons to fail, even with good intentions…

- Inadequate bridge between business and technology. You haven't really addressed who should extend the model and how.
- Poor communication between domain experts and programmers.
- You underestimate or don't provide good support for operations and deployment.
- Your domain experts aren't good modelers.

Meta Collaborators

- Ademar Aguiar
- Francis Anderson
- Ali Arsanjani
- Jean Bezivin
- Paulo Borba
- Filipe Correia
- Krzysztof Czarnecki
- Ayla Dantas
- Martine Devos
- Hugo Ferreira
- Brian Foote
- Martin Fowler
- Richard Gabriel
- Eduardo Guerra
- Fabio Kon
- Atzmon Hen-Tov
- Ralph Johnson
- David H. Lorenz
- Patricia Matsumoto
- Lena Nikolaev
- Jeff Oaks
- Reza Razavi
- Nicolas Revault
- Dirk Riehle
- Lior Schachter
- Dave Thomas
- Michel Tilman
- Leon Welicki
- Rebecca Wirfs-Brock
- …



Conversa com especialistas

Como construir Software Extraordinário

Este é um evento gratuito, PRESENCIAL E ONLINE

DATA DO EVENTO:
03/10/2022
DAS 17H30 ÀS 20H30

Público destinado: estudantes de programação, pessoas engenheiras de software, POs, gerentes de produto e agilistas.

Organizado e apoiado por:

ITURING
IME INSTITUTO DE MATEMÁTICA E ESTATÍSTICA UNIVERSIDADE DE SÃO PAULO
Refactory

tinyurl.com/creating-great-software

Marden Neubert
Technology Advisor no PagBank

Mariana Martins
Especialista Ágil na Bain & Company

Graziela Tonin
Professora Dr. no Insper e consultora em metologias ágeis

Alfredo Goldman
Professor Dr. na USP

https://conteudo.ituring.com.br/evento-como-criar-software-extraordinario

Muito Obrigado!!!

yodamann

joe@refactory.com

@metayoda

Slides available at: https://refactory.com/papers/USP-MetaArchitecture.pdf

“Anything you can do, I can do meta ;-)”
“If you think good architecture is expensive, try bad architecture"