Architecture and Design of Adaptive Object-Models

Joseph W. Yoder

&

Ralph Johnson

The Refactory, Inc.

University of Illinois

yoder@refactory.com

johnson@cs.uiuc.edu

http://www.adaptiveobjectmodel.com

Presenter

- Joseph Yoder
  - e-mail: yoder@refactory.com
  - www: http://www.joeyoder.com

  - www.refactory.com

Table of Contents

What are we doing?

- Describing a "new" type of architecture.
- Not really "new" but only known by a few people. Has not been well described.
- Going to describe them with patterns.
- Patterns work together to solve the problem.

Why is this Important?

- Need to know different architecture styles so as to understand them, know the advantages and disadvantages, when to use them, etc.
- Software architects usually reuse architectures they understand.
- Learn through courses, experience, working with someone else, papers, …
- Need to know and catalogue and describe in detail different types of architectures.

Meta Collaborators

- Ali Arsanjani
- Paulo Borba
- Krzysztof Czarnecki
- Ayla Dantas
- Martine Devos
- Brian Foote
- Martin Fowler
- Ralph Johnson
- Jeff Oaks
- Nicolas Revault
- Dirk Riehle
- Reza Razavi
- Michel Tilman
- Dave Thomas
- Others...

Adaptive Object-Models

- Architectures that can dynamically adapt to new user requirements by storing descriptive (metadata) information about the business rules that are interpreted at runtime.
- Sometimes called a "reflective architecture" or a "meta-architecture ".
- Highly Flexible – Business people (non-programmers) can change it too.

Context of the Style

- Requirements change within application's domain.
- Business Rules are changing rapidly.
- Applications have to quickly adapt to new business requirements.
- Changing the application is costly, it generally includes code and data-storage.
- There are cycles of: build-compile-release.

Forces – Shearing Layers

- Who (Business Person, Analyst, Developer)
- What (Business Rule, Persistence Layer,…)
- When (How often, How fast)

There is a different rate of change on the system.

Foote & Yoder - Ball of Mud PLoPD4

General Idea

- Create an object design (meta-model) that describes the domain objects which includes attributes, relationships, and business rules as instances rather than classes.
- The domain objects are instantiated through a description given by the user or domain expert.
- Each new requirement is satisfied by creating a new description and a new instantiation.
- Separate what changes from what doesn't.
- Define Changes without Hand-Coding.
- Focus on "What" Not "How".

Adaptive Object-Model

- An ADAPTIVE OBJECT-MODEL is an object model that provides "meta" information about itself so that it can be changed at runtime
  - explicit object model that it interprets at run-time
  - change object model, the system changes its behavior
- ADAPTIVE OBJECT-MODELS usually arise as domain-specific frameworks
- Business rules are stored as descriptive (meta) information in ADAPTIVE OBJECT-MODELS

Adaptive Object-Models

- Represents classes, attributes, relationships, and behavior as *metadata*.
- Based on instances rather than classes.
- Users change the *metadata* (object model) to reflect changes in the domain.
- Stores its *Object-Model* in a database or in files and interprets it (can be XML/XMI).

Consequently, the object model is adaptable, when you change it, the system changes immediately.

Introduction of Metadata and Adaptive (Active|Dynamic) Object-Models

"Anything you can do, I can do Meta"

Metadata: 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"

Code is Data, Data is Code – Everything is Data

Architectural Elements of Adaptive Object Models

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

If you want something to change quickly,
you must push it into the data.

TypeObject: Problem

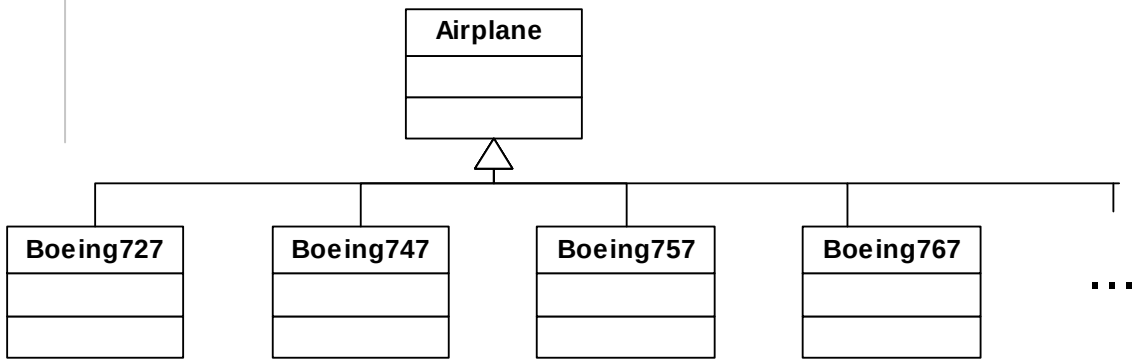
- Making a class representing similar types of information will create structural and data duplication.
- Making many different classes can make any system hard to maintain; each time a new class is added the system should be updated with the new class and the database updated (new releases).

- *There are many instances of a particular kind of element in the domain with minor differences*

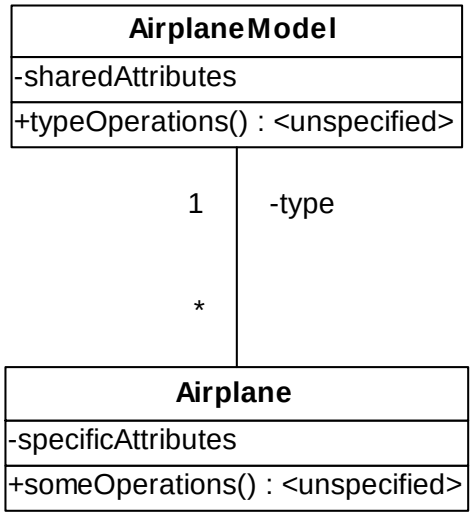
TypeObject: Solution

PLoPD3 - Johnson and Woolf

Before



After

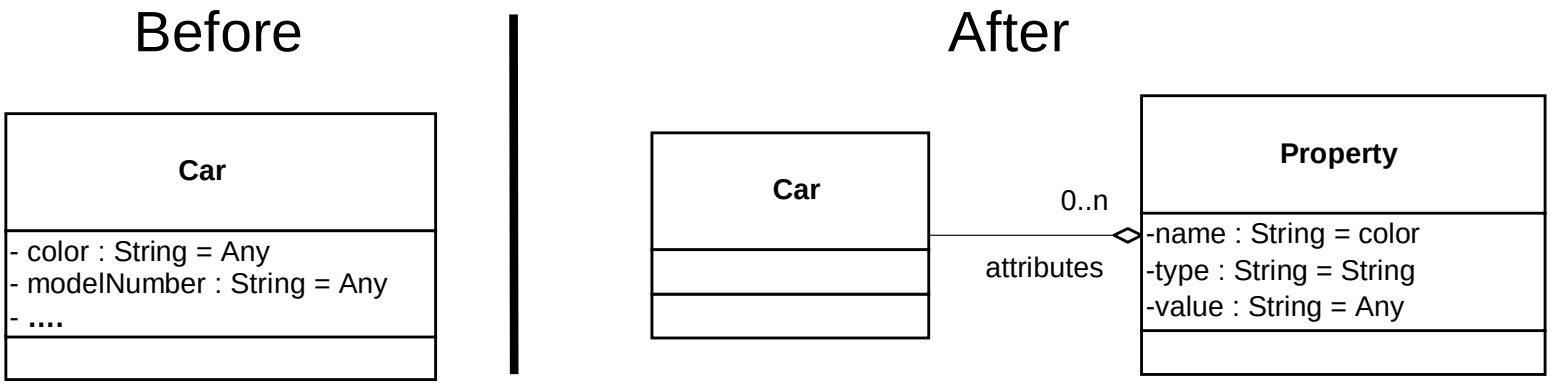


Properties: Problem

- Making subclasses based on their attributes makes the system static in nature. Each time a new attribute is required, a new class is created, and the system updated.
- The model will have a prolific hierarchy of classes representing the same domain abstraction or many versions that need to be released to represent the differences.

- *Instances of a given class might have different attributes that may vary at runtime.*

Properties: Solution



Example: A Store with Catalogue Entries

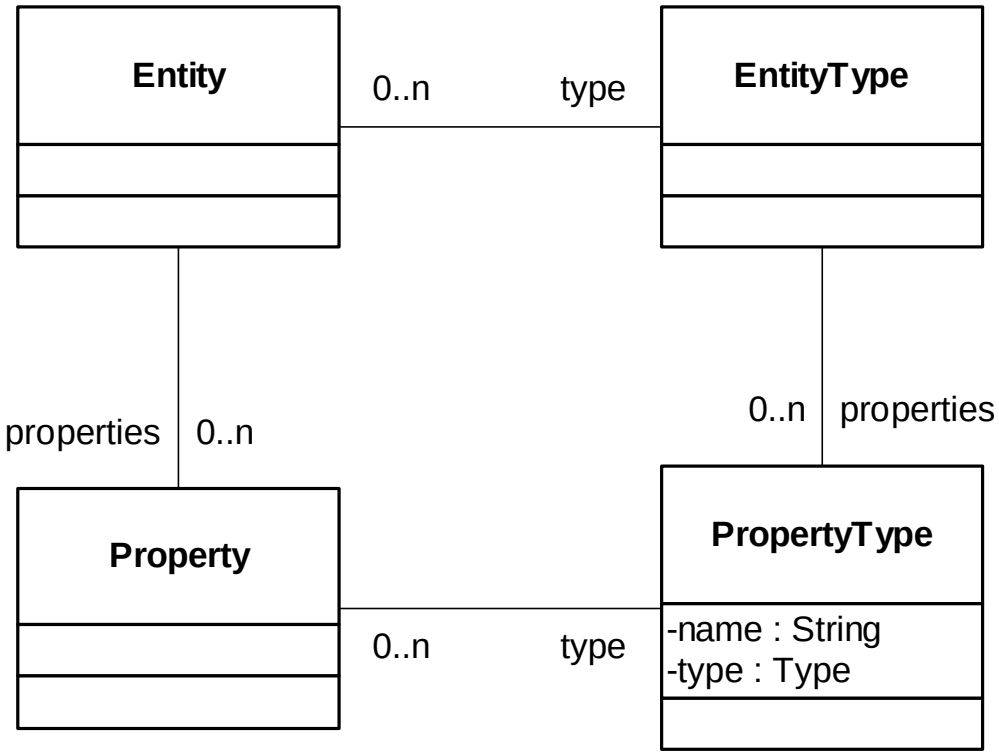
- Sweaters (size, color, material)
- Canoes (length, material)
- Video Tapes (name, rating, category)

TypeSquare: Problem

- Fixing the property types information for each property forces the system to be changed each time a property is added or changed on its type.
- The system still need to be able to entirely define new high level Types of Properties.

- *The system needs to handle types on dynamic properties and you want to ensure that the types of properties are correct.*

TypeSquare: Solution



Example: A Store with Catalogue Entries

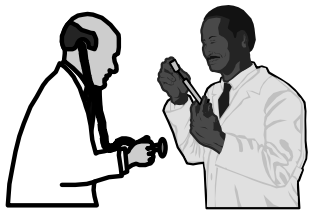
- Sweaters (size=(S,M,L,XL), color=(red,green,blue,yellow,…)
- Canoes (length=float, width=float)

Entity-Relationship: Problem

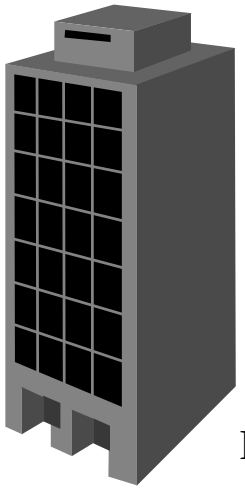
- Elements from the domain can have multiple relationships between them and the system may need to keep track of the relationships including time and other relevant information.

- *The application needs to expose the relationships between domain elements (entities). Each relationship gives roles to the participants. Roles can change and the system needs to know what current roles are being played and are available.*

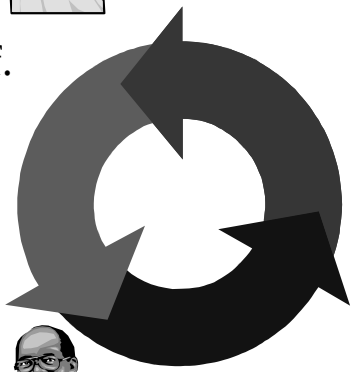
Newborn Screening



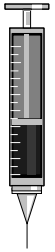
Doctor, HealthProf.



Hospital, Lab



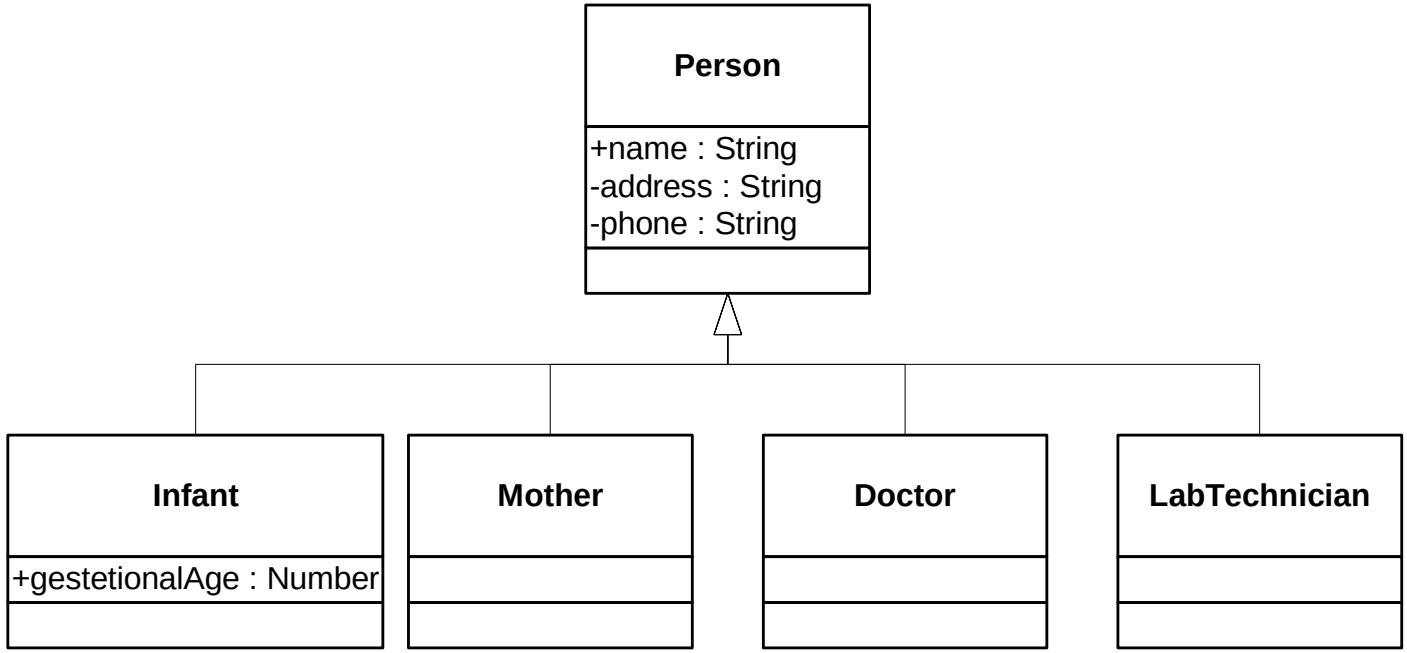
Mother, Infant



Blood Specimen

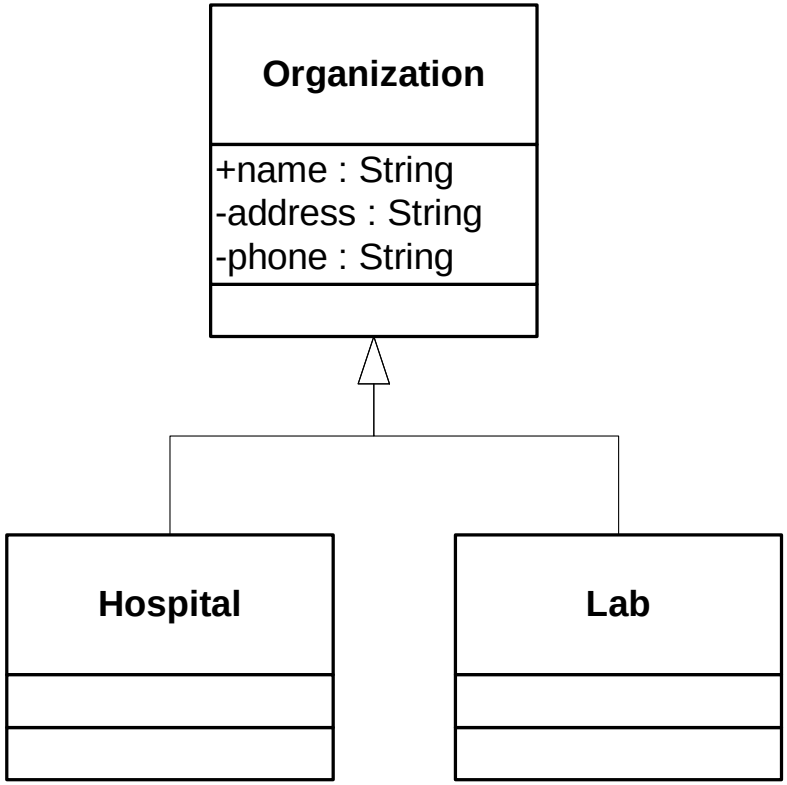
Newborn Screening

Infants, Mothers and Doctors...



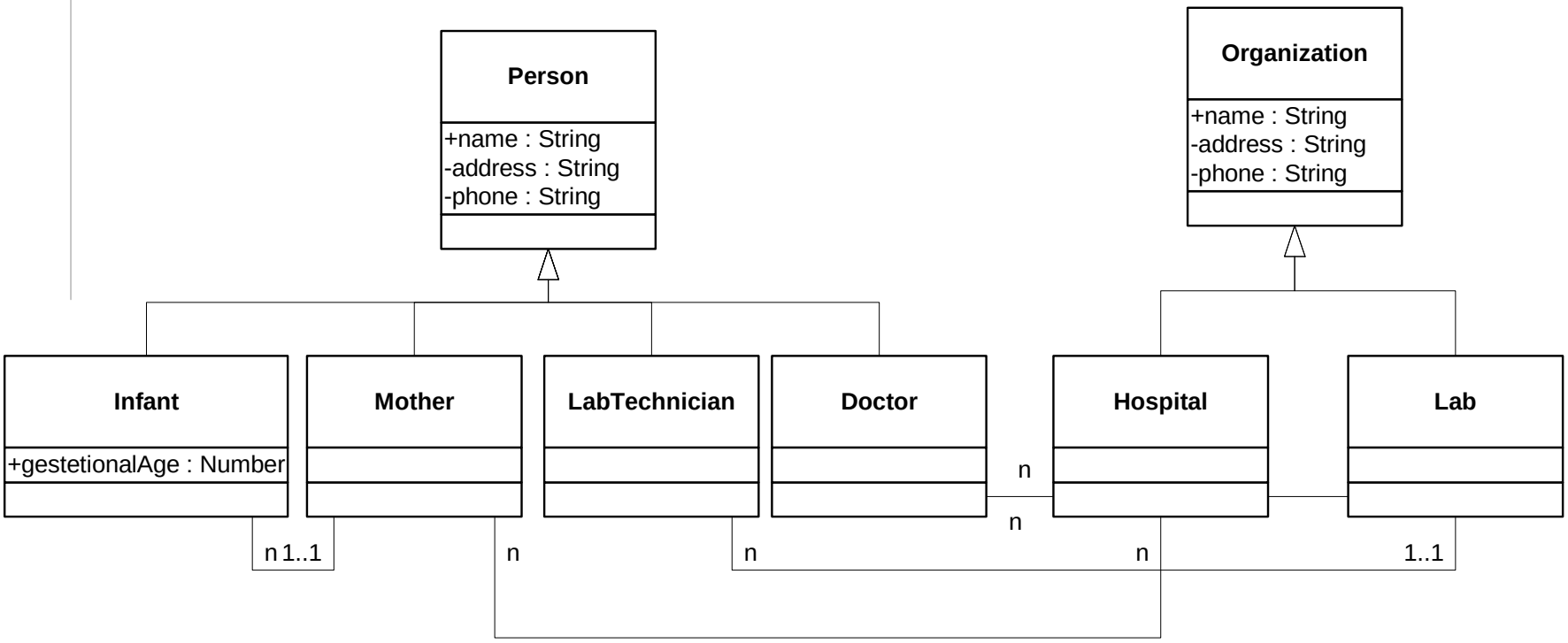
Newborn Screening

Hospital and Lab

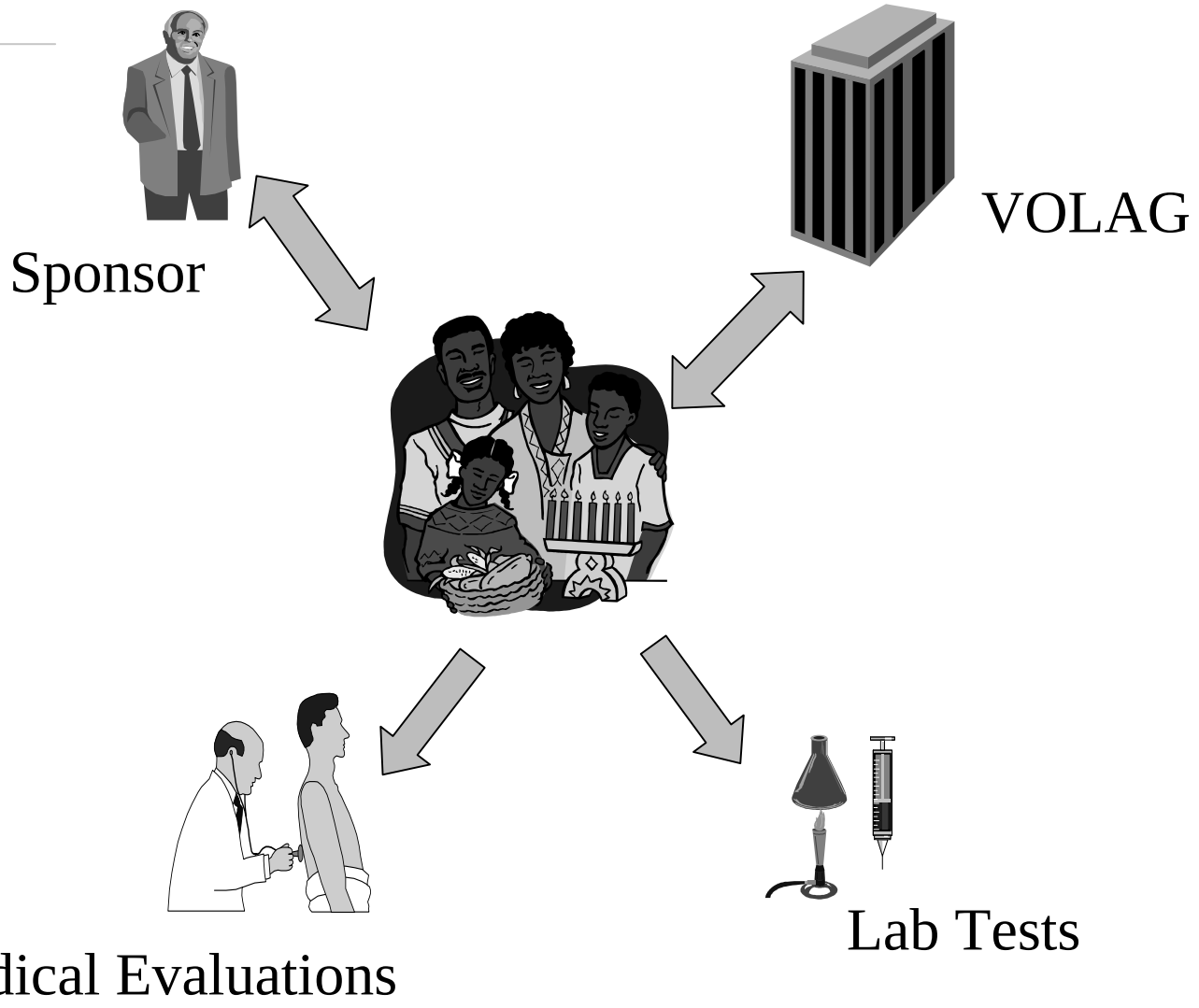


Newborn Screening

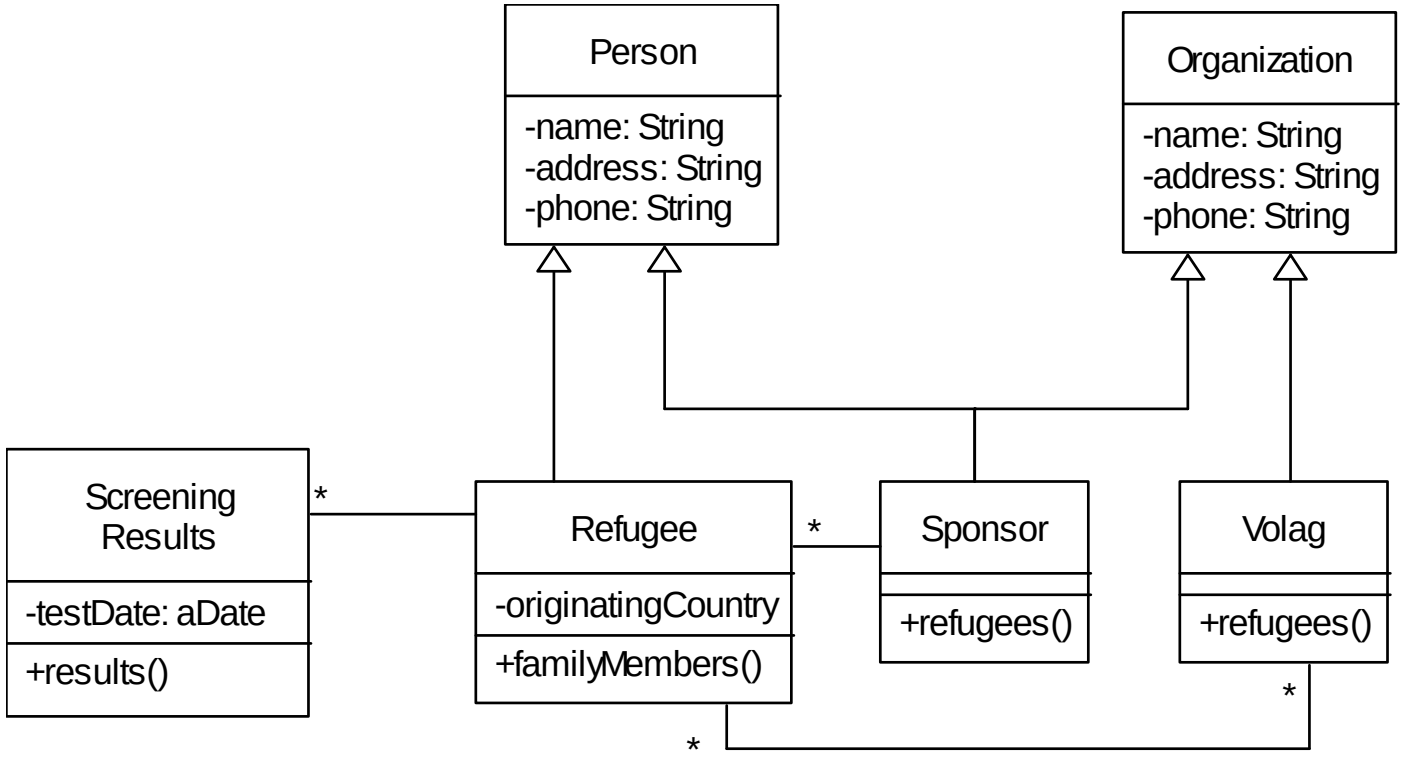
Putting it all together



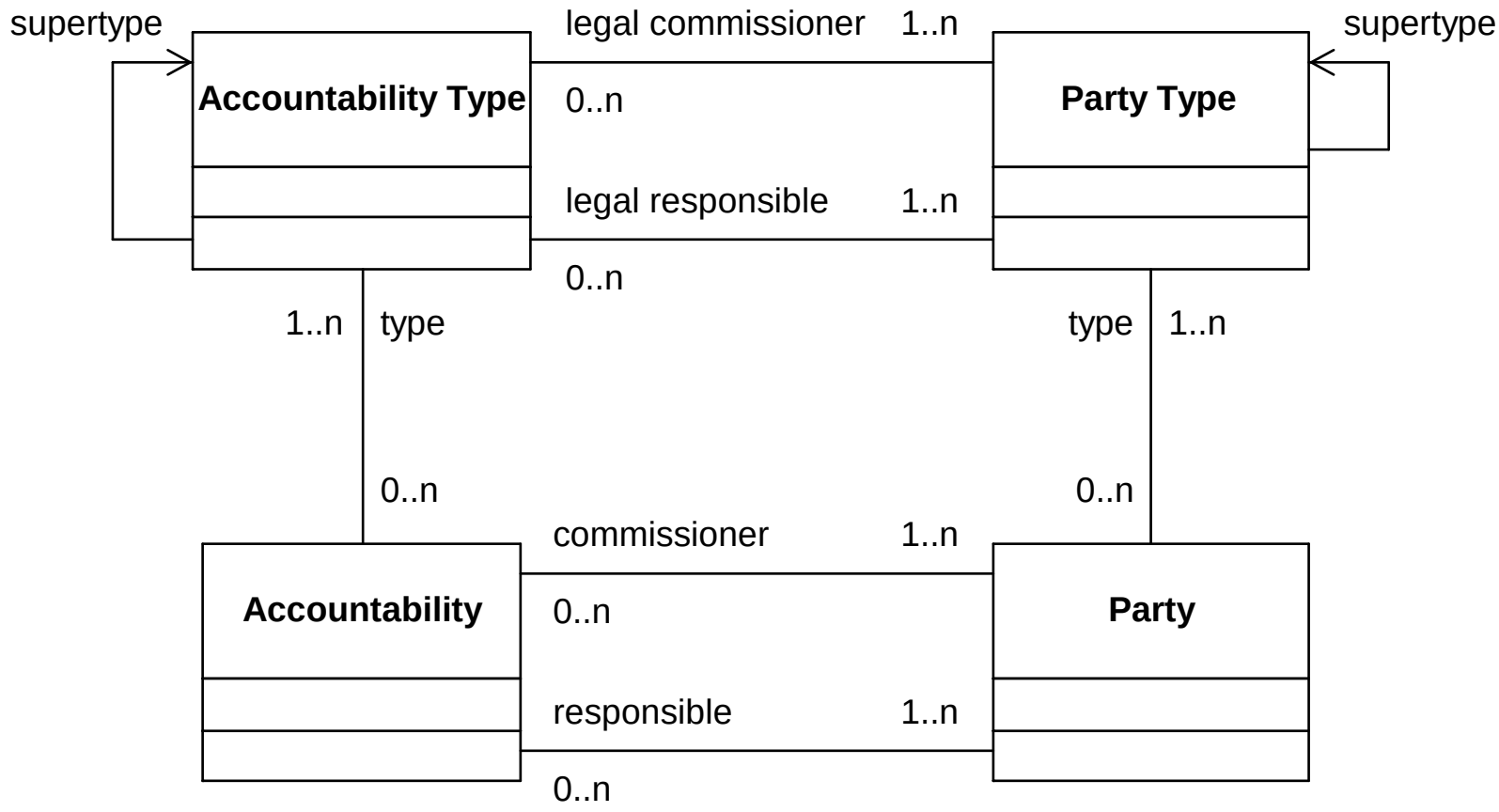
Refugee



Refugee



Entity-Relationship: Solution



Analysis Patterns - Fowler

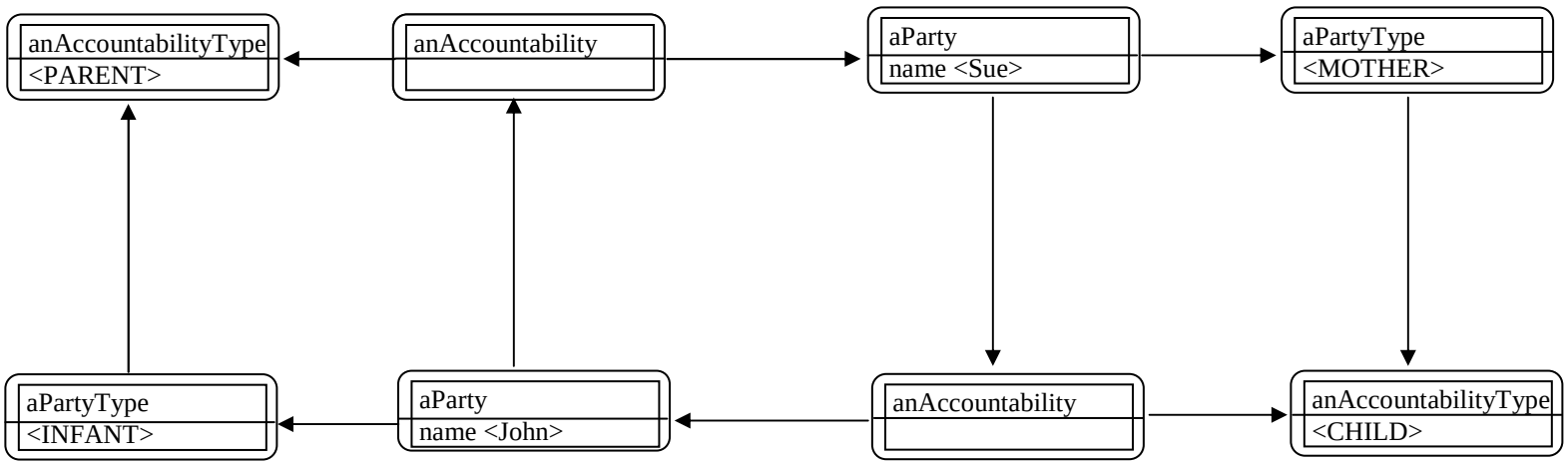
Party and Accountability
Example

Sue Smith

John Smith

Sue is the mother of John

Party and Accountability
(instance diagram)

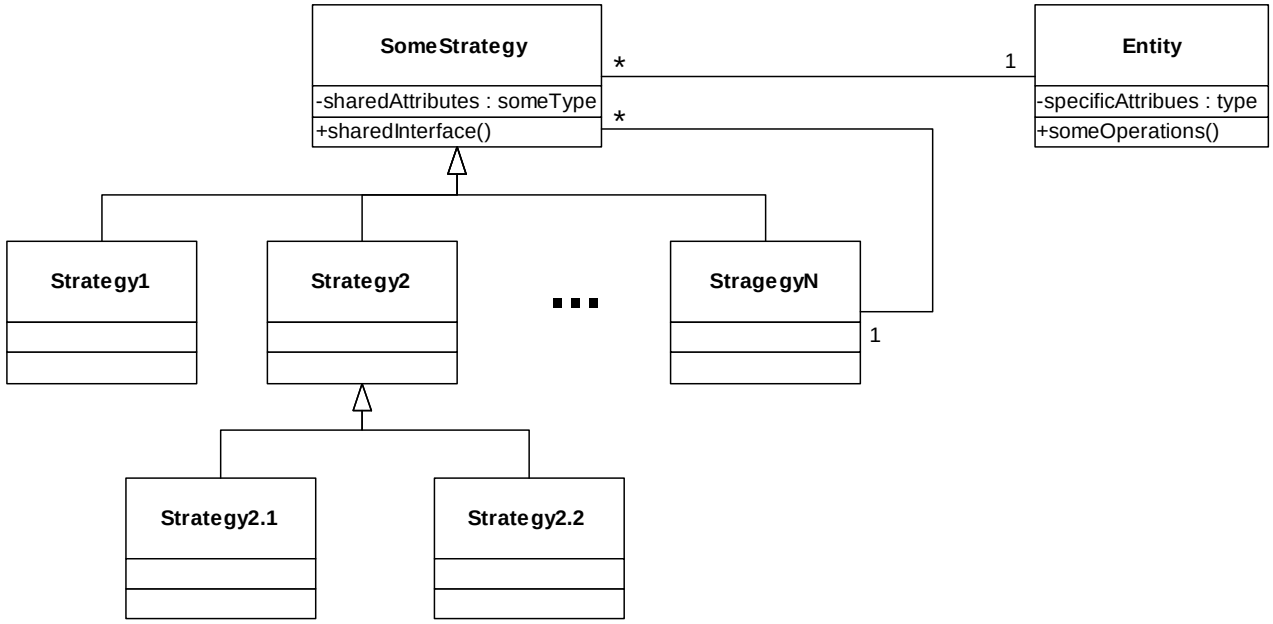


Strategy: Problem

- 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.*

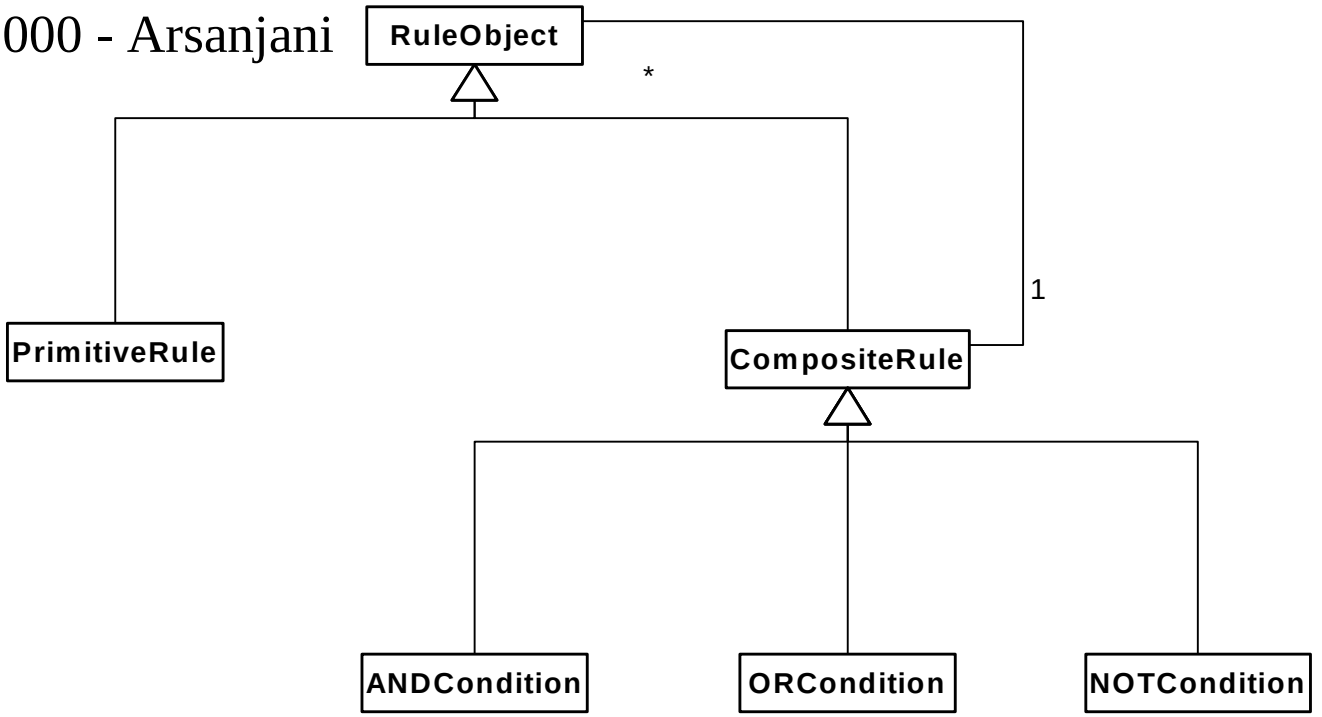
Strategies/RuleObjects Solution (Behavior/Methods)



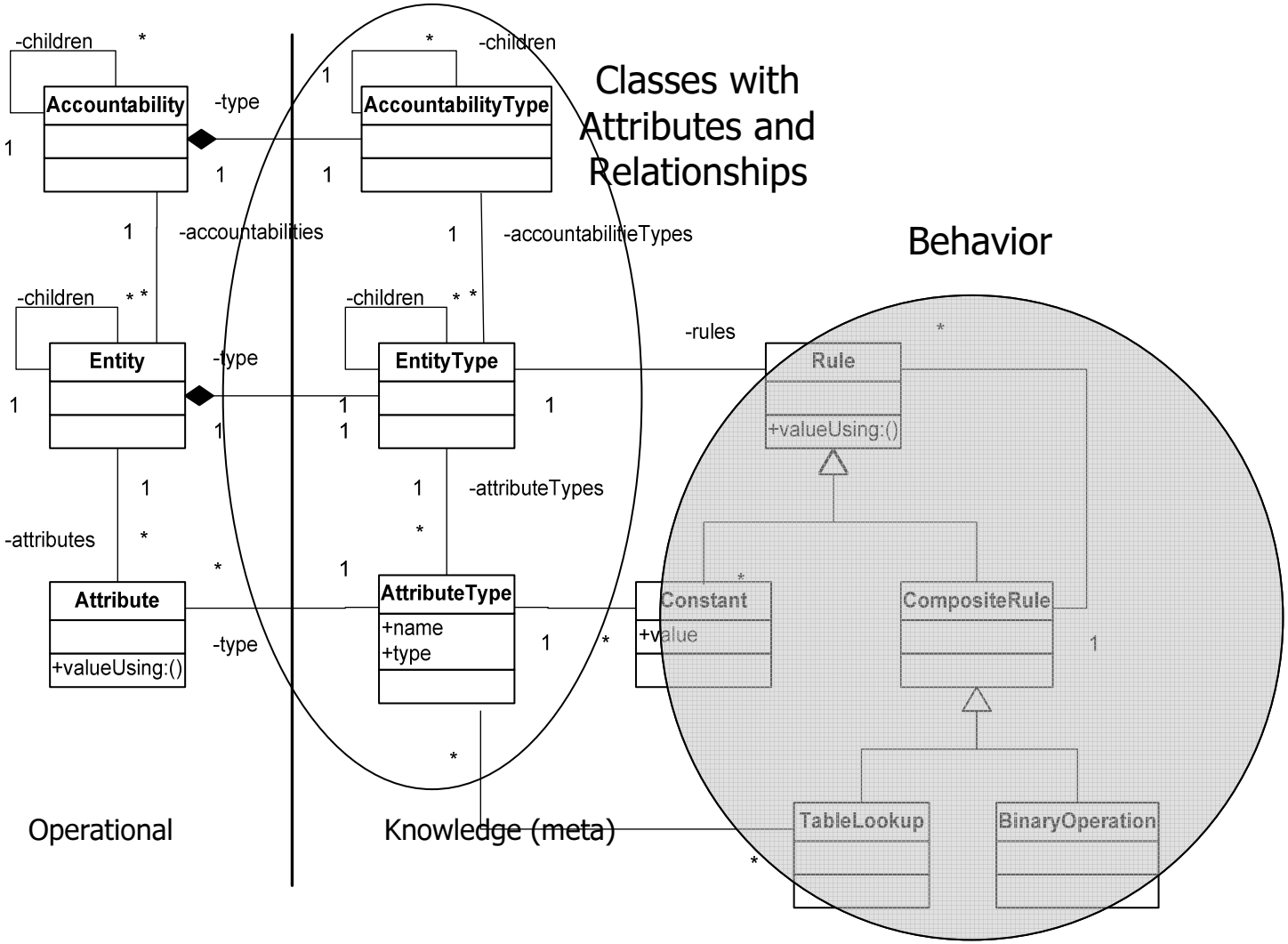
Design Patterns - GOF95

RuleObject: Configurable Strategies

PLoP 2000 - Arsanjani



Adaptive Object Model "Common Architecture"



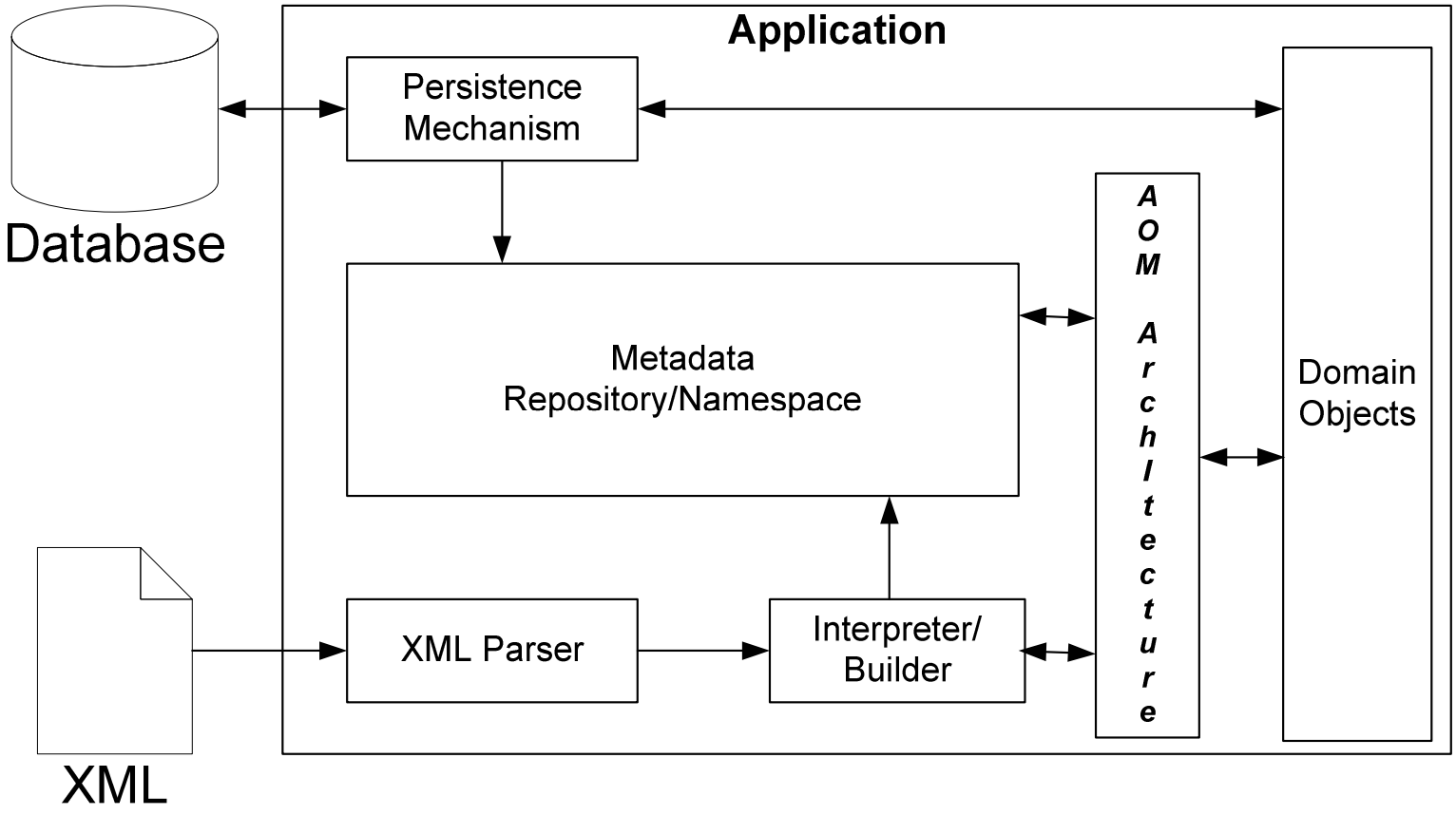
Interpreters / Builders: Context

- *Adaptive Object-Models need to implement ways for describing the types of entities, properties, and relationships as well as ways to create them from this description.*
- *The system also needs a way to interpret the dynamic behavior at runtime such as the strategies or rules.*

Interpreters / Builders: Problem

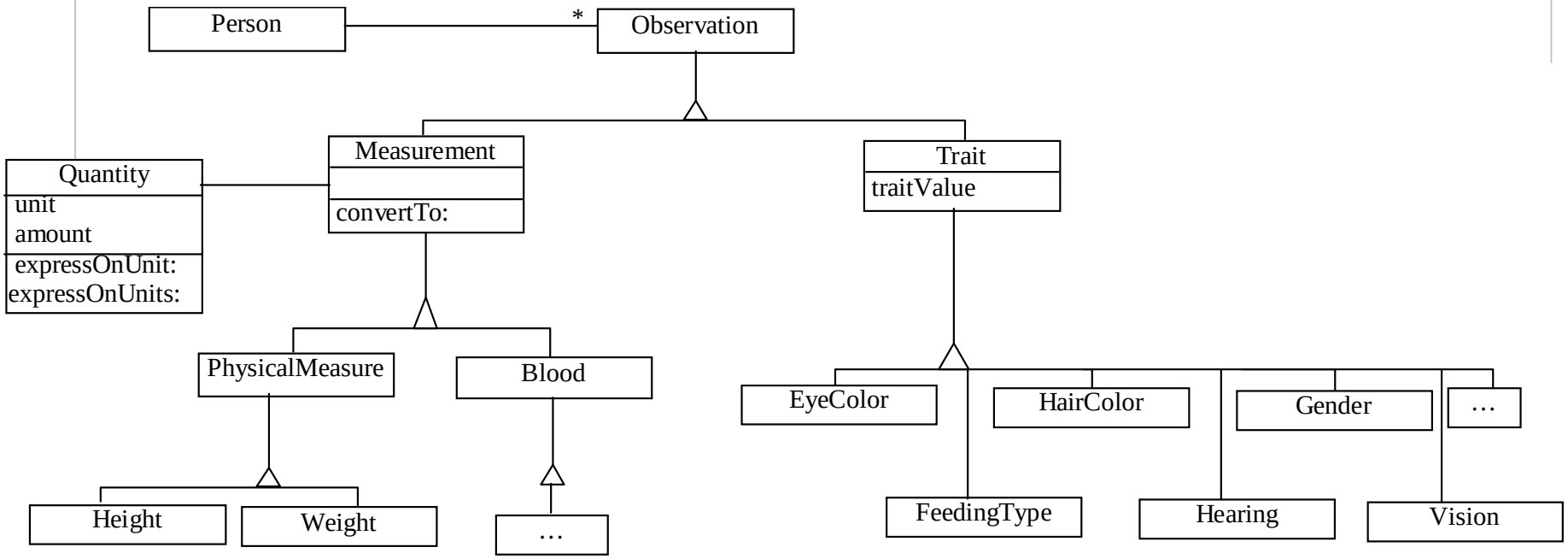
- Creating methods in each class for instantiating the required metadata defeats the purpose of taking the AOM approach.
- The system has to be able to read the metadata any time, and configure itself.
- The metadata is based on the knowledge of domain experts rather than developers.
- Rules are based upon descriptive (meta) information

Interpreters / Builders "Virtual Machine"



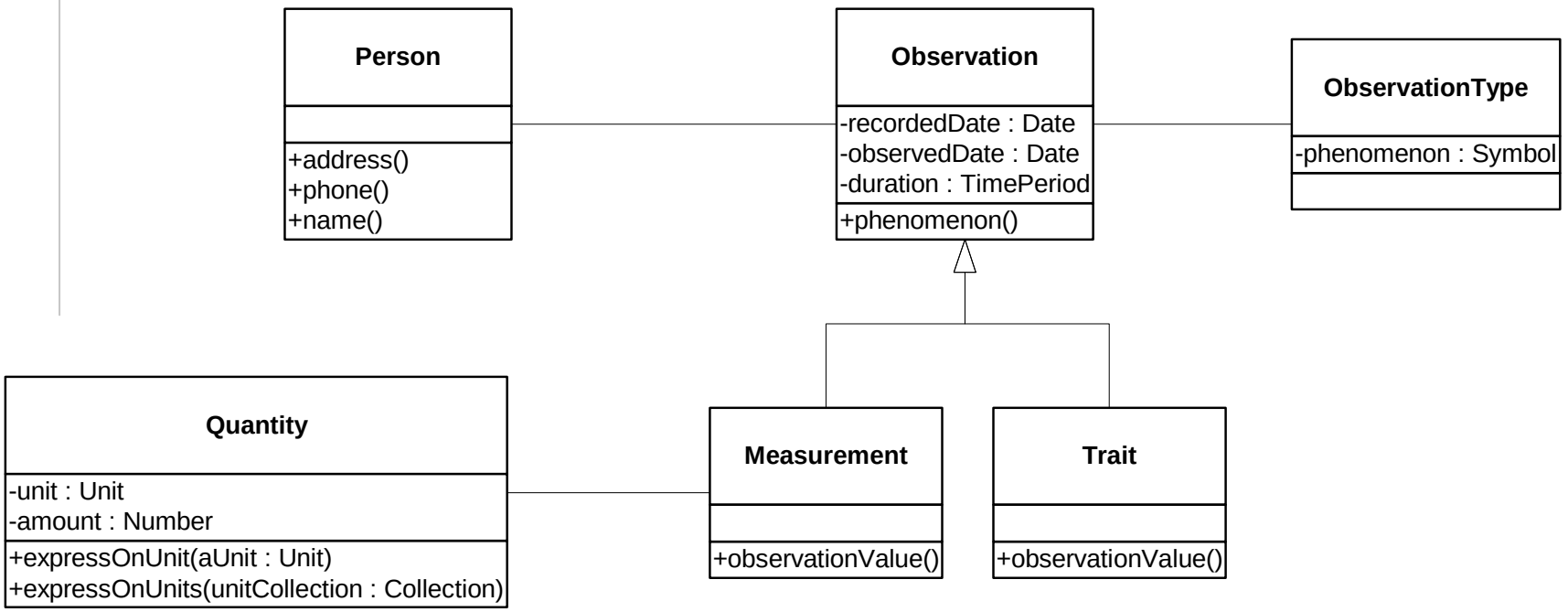
Adaptive Object-Model Examples

Medical Observation - First Model



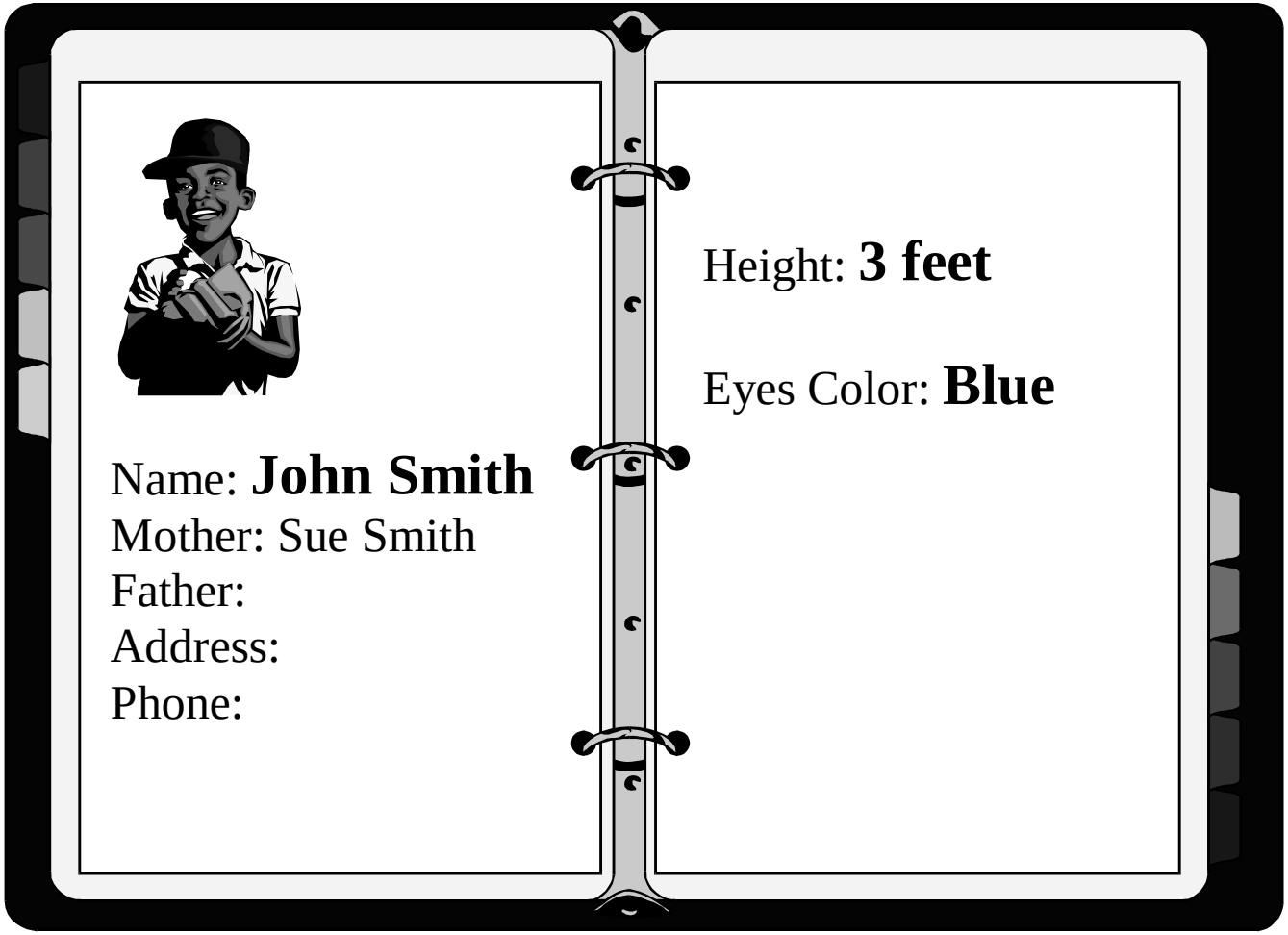
What happens when a new observation is required?

Observation Design

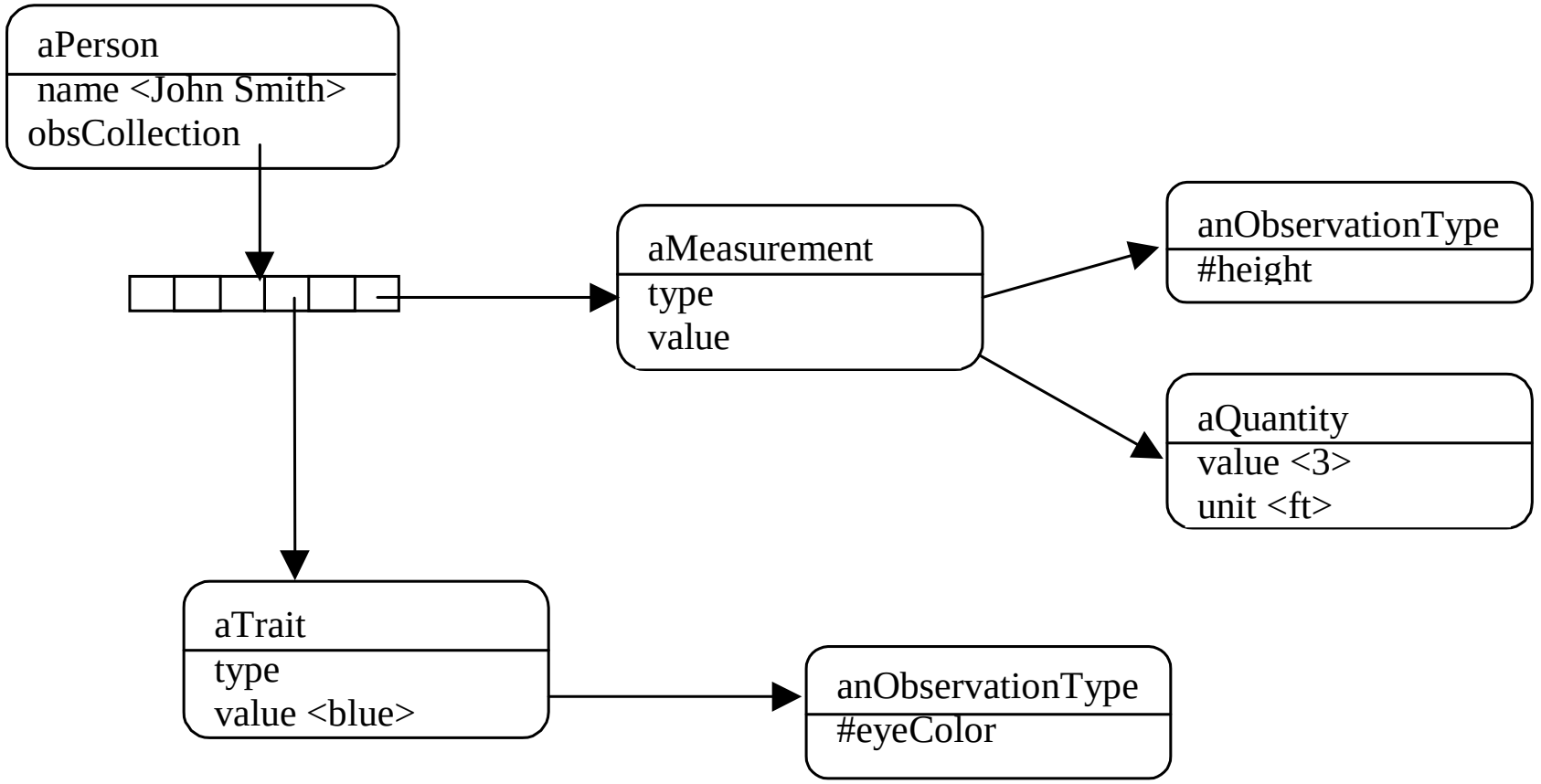


Observation Design

Example



Observation Design
(instance diagram)

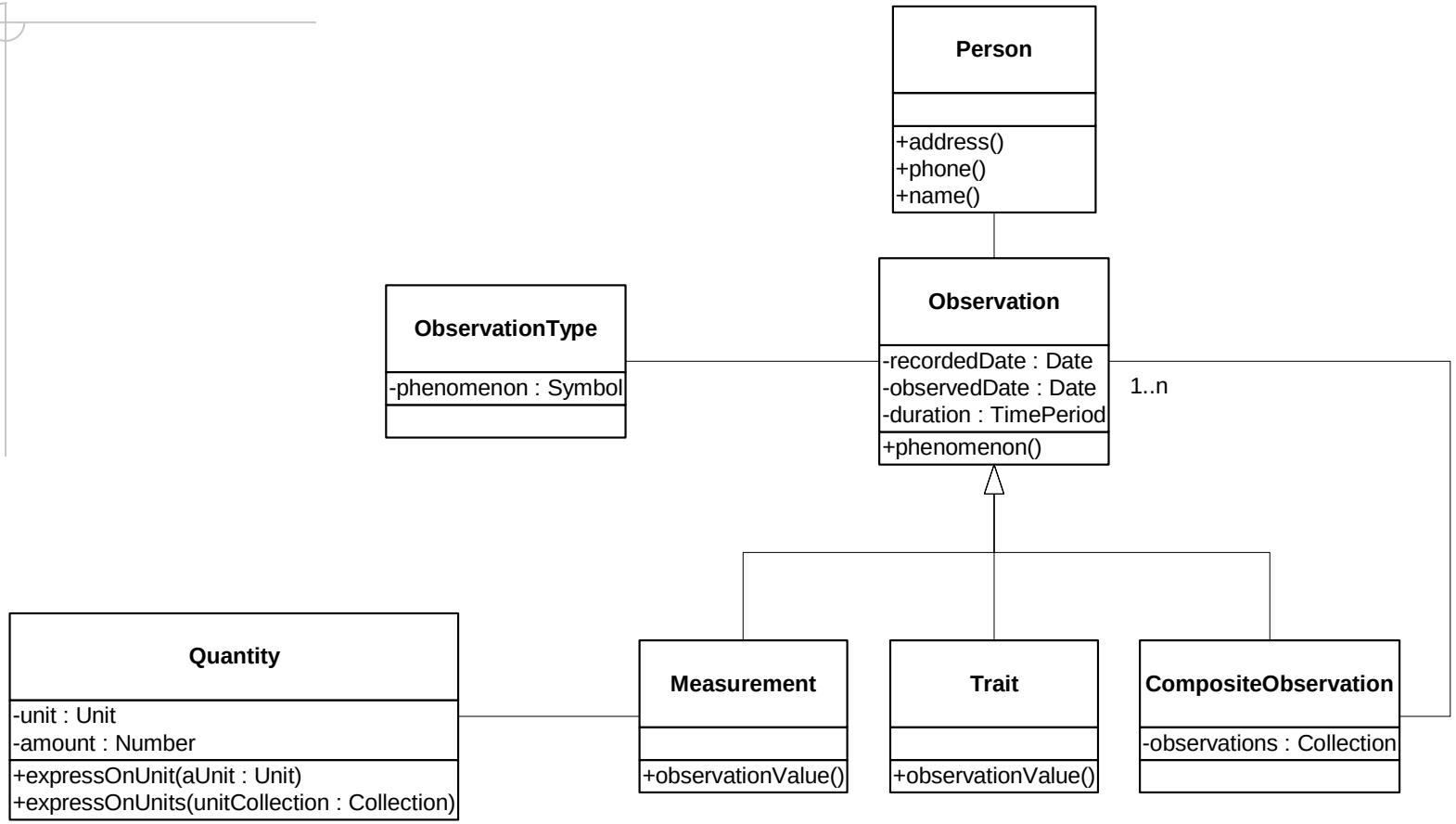


Composing Observations

Observations can be more complex

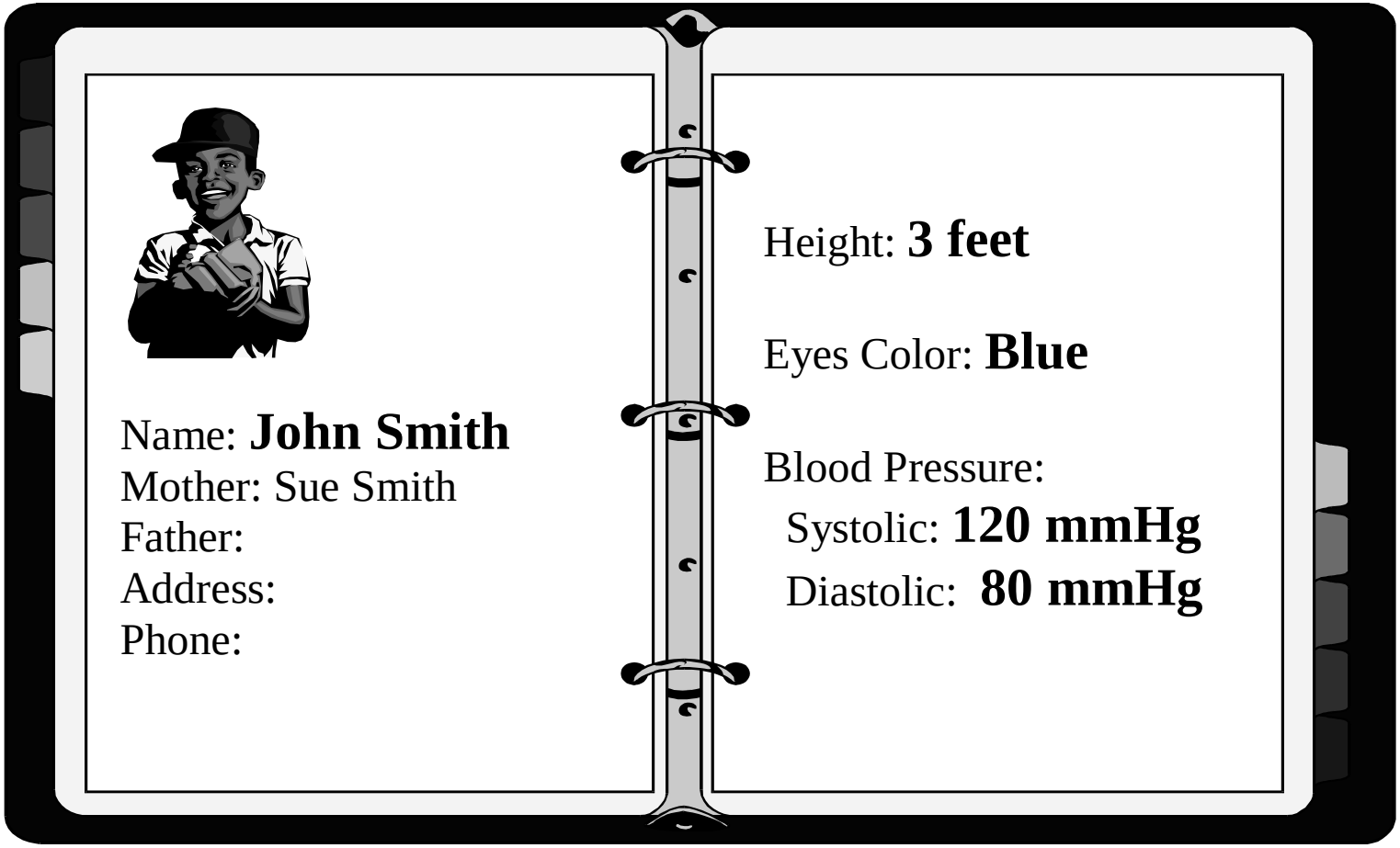
- Cholesterol
  - Components: HDL, LDL
- Blood Pressure
  - Components: Systolic, Diastolic
- Vision
  - Components: Left Eye, Right Eye

Composite Observation Design

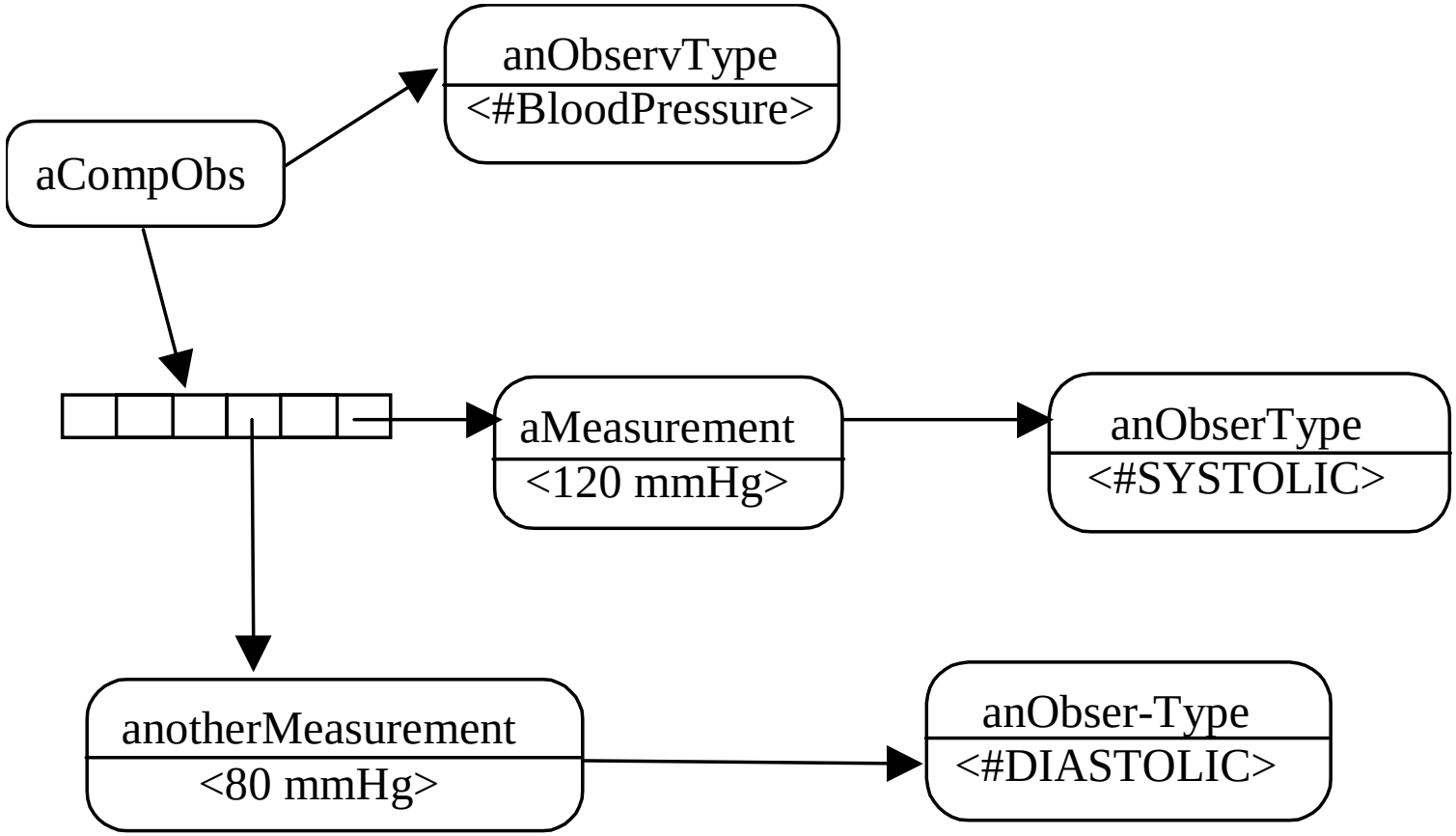


Composite Pattern (GOF)

Observation Design
Example

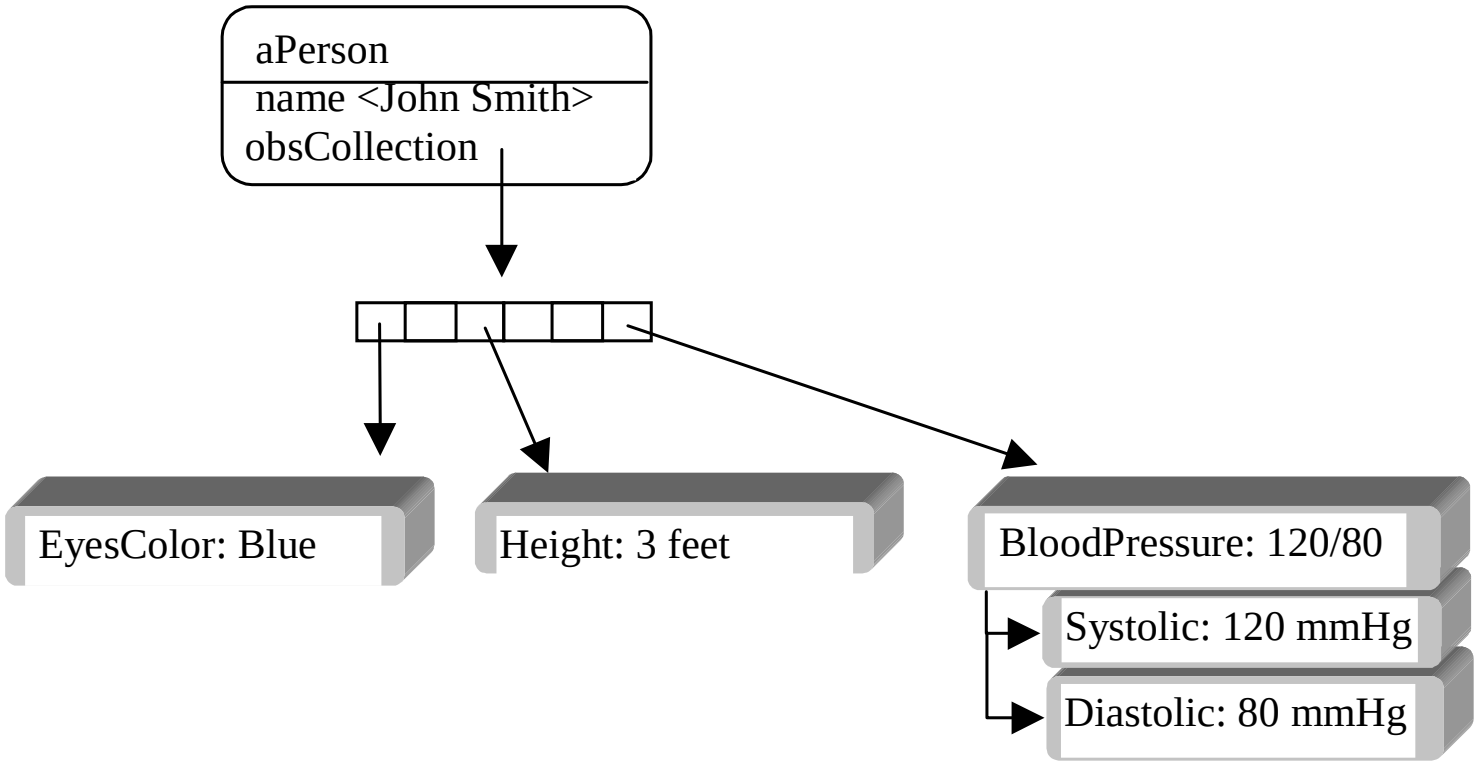


Composite Observation Design
(instance diagram)



Composite and Primitive Observation Design

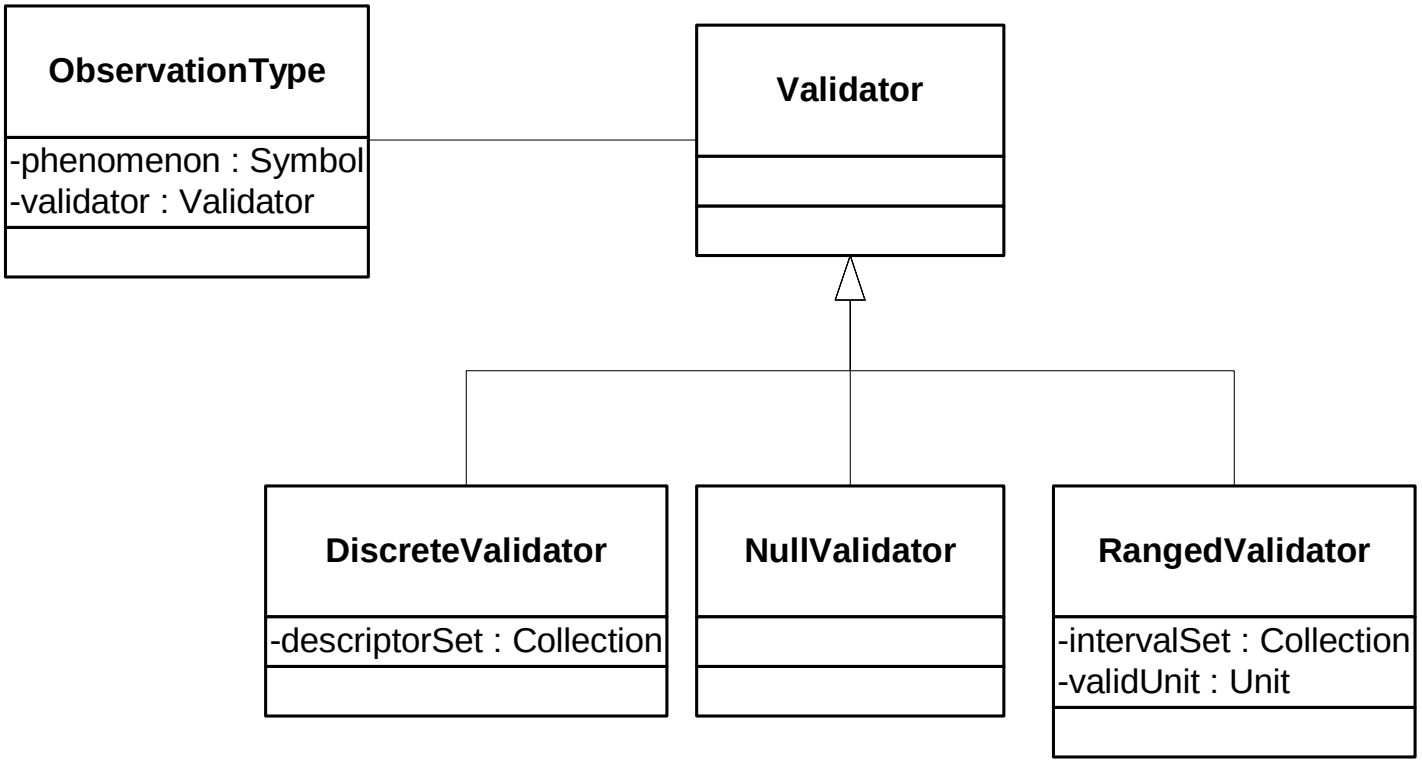
What we know about John?



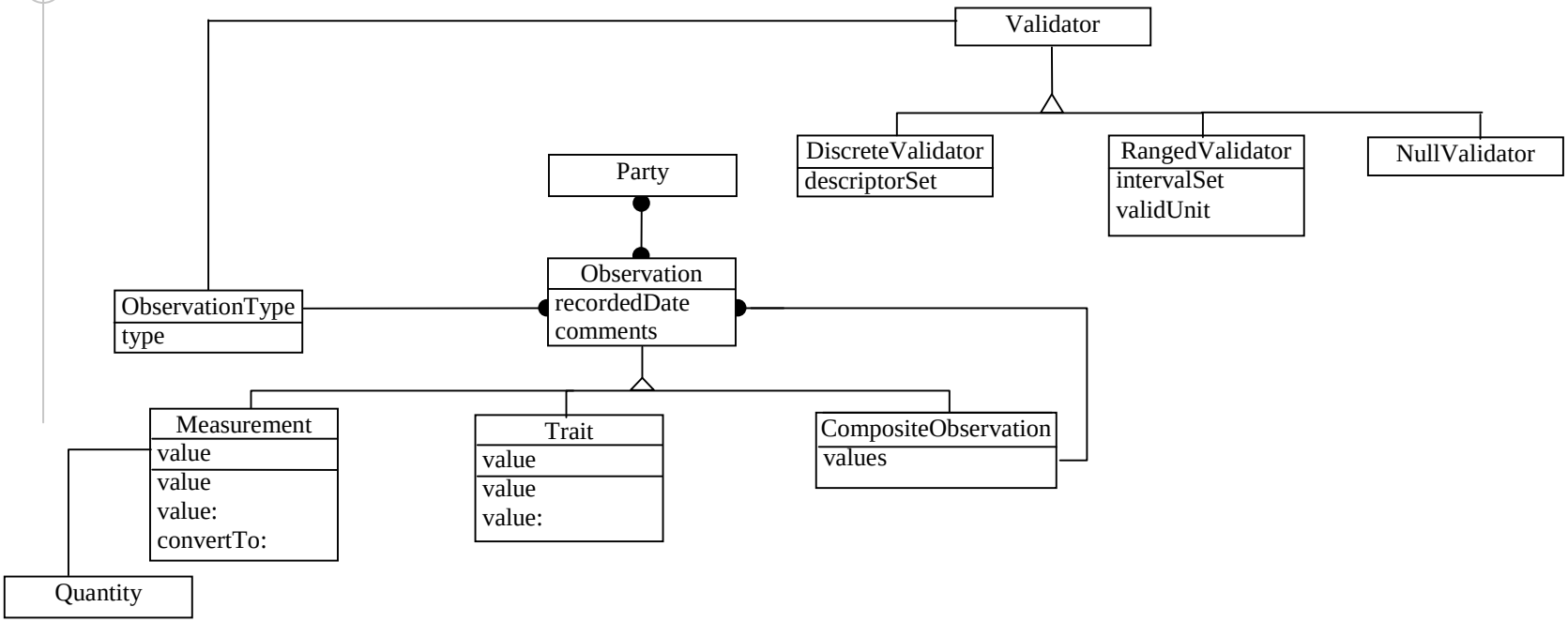
Validating Observations

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

Validating Observation Design



Overall Framework's Design

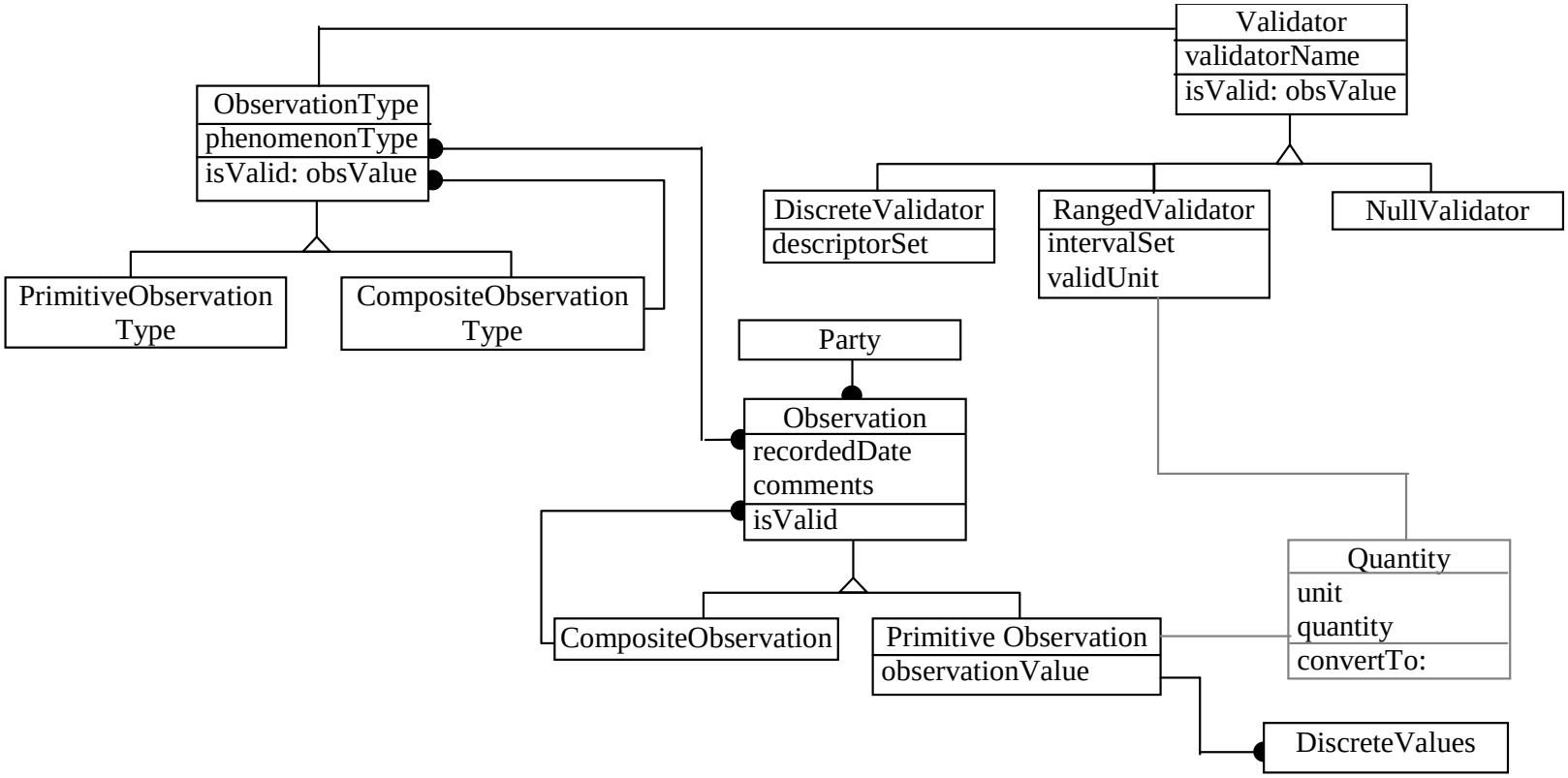


Is everything an Observation?

How does the model specify the structure of the Composite?

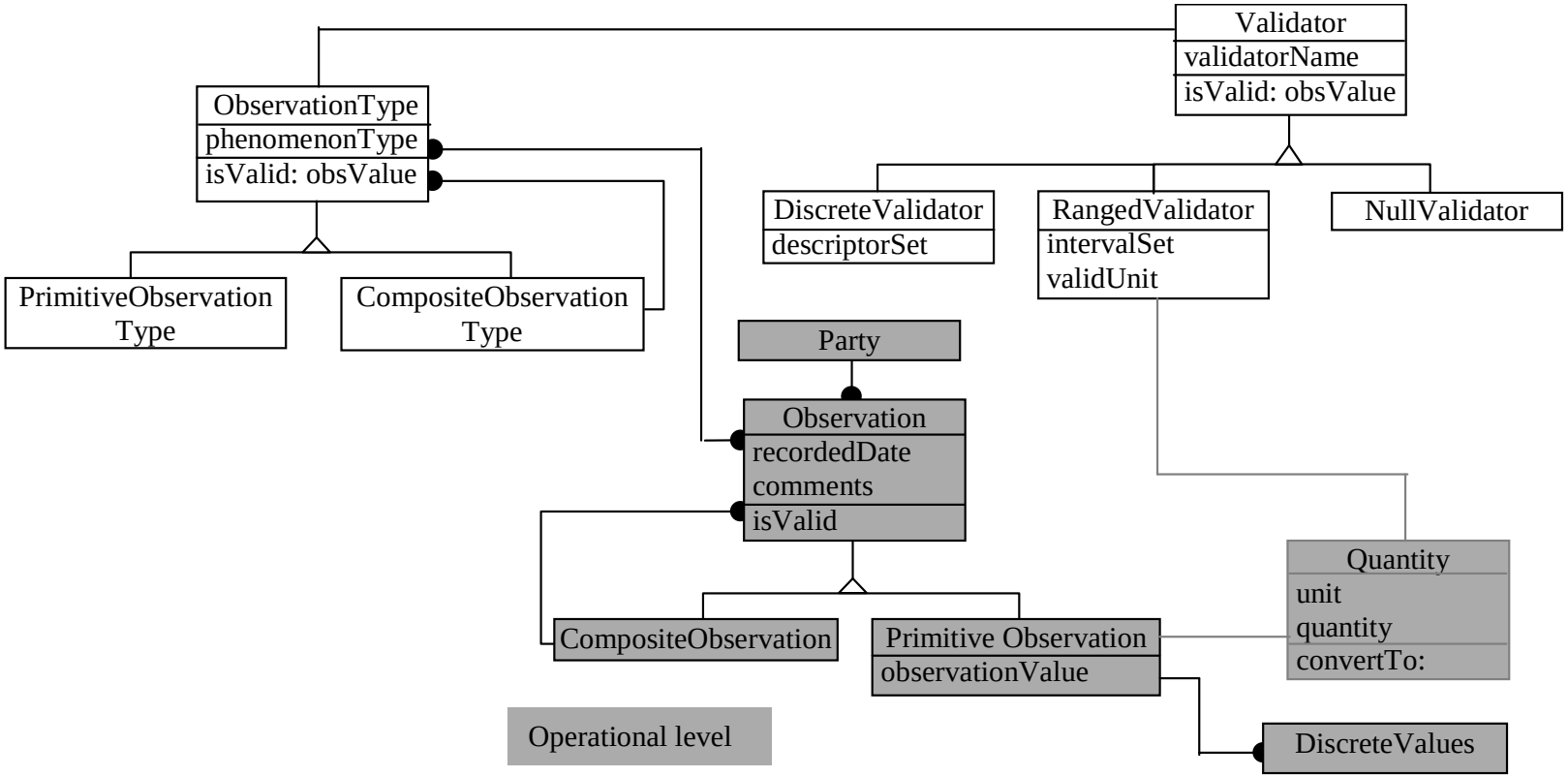
What is the relationship between Trait and DiscreteValidator?

Framework's Design



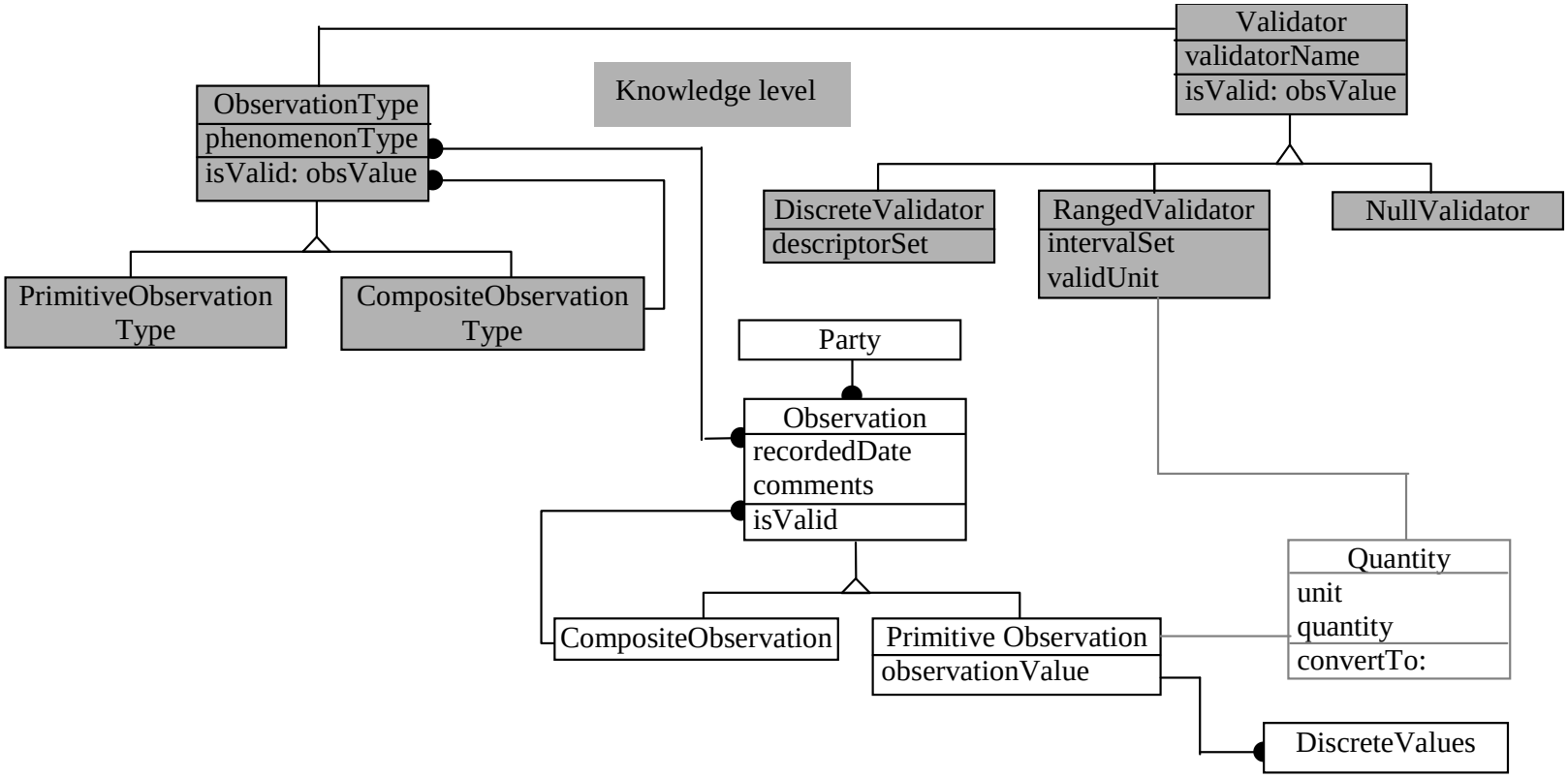
Current Design

Framework's Design



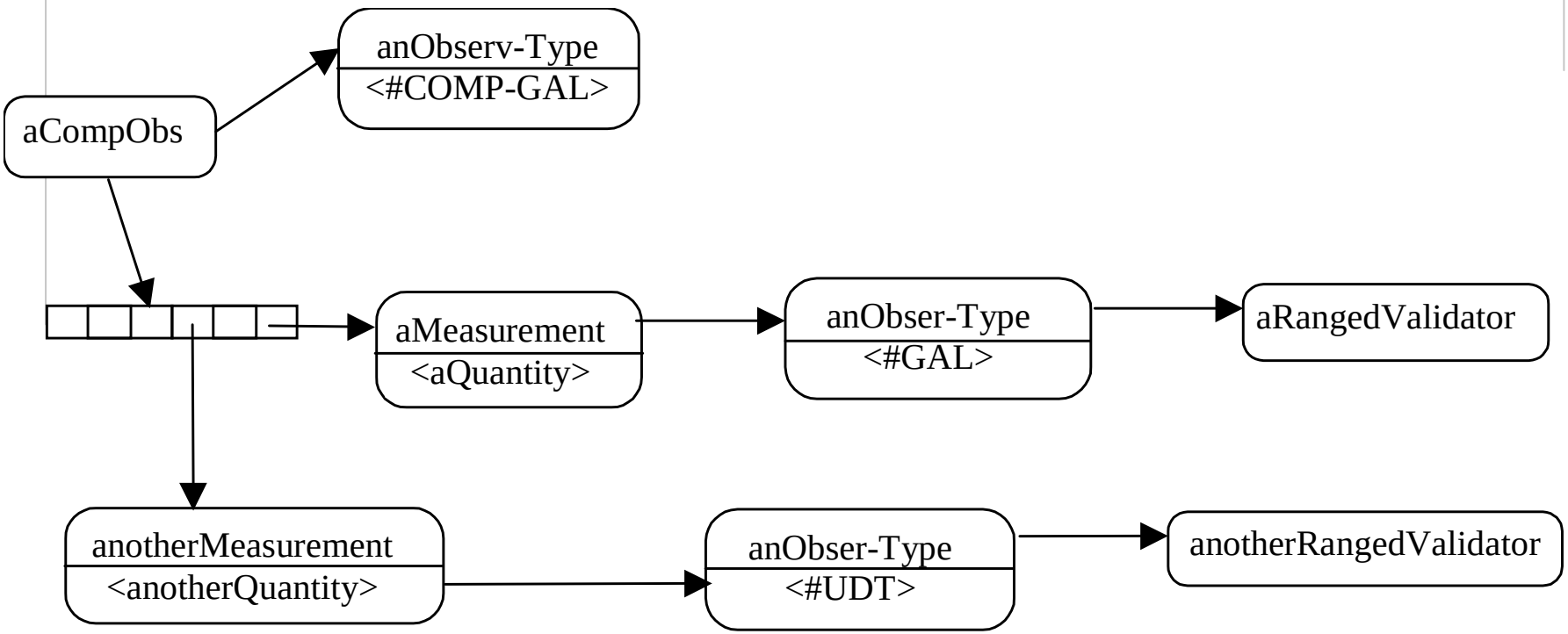
Current Design

Framework's Design

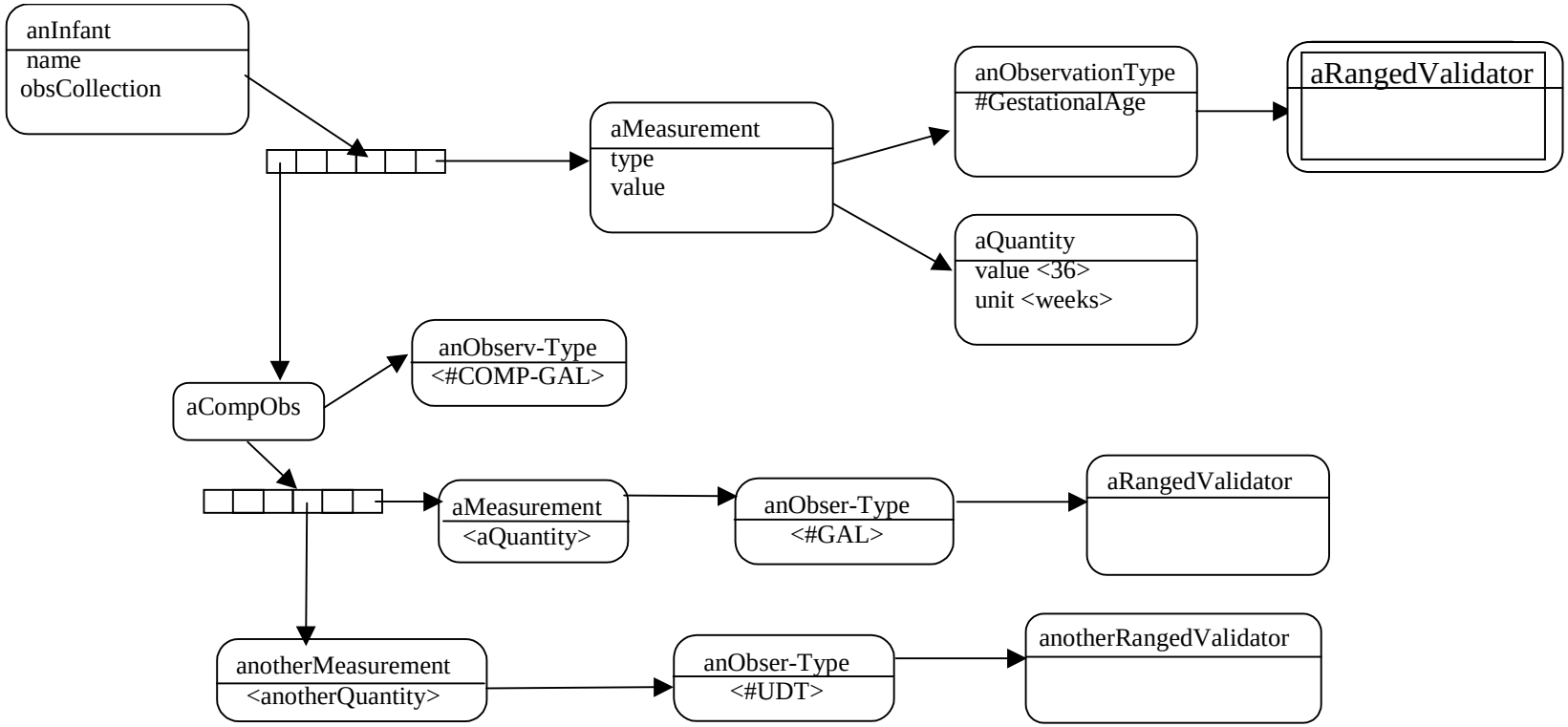


Current Design

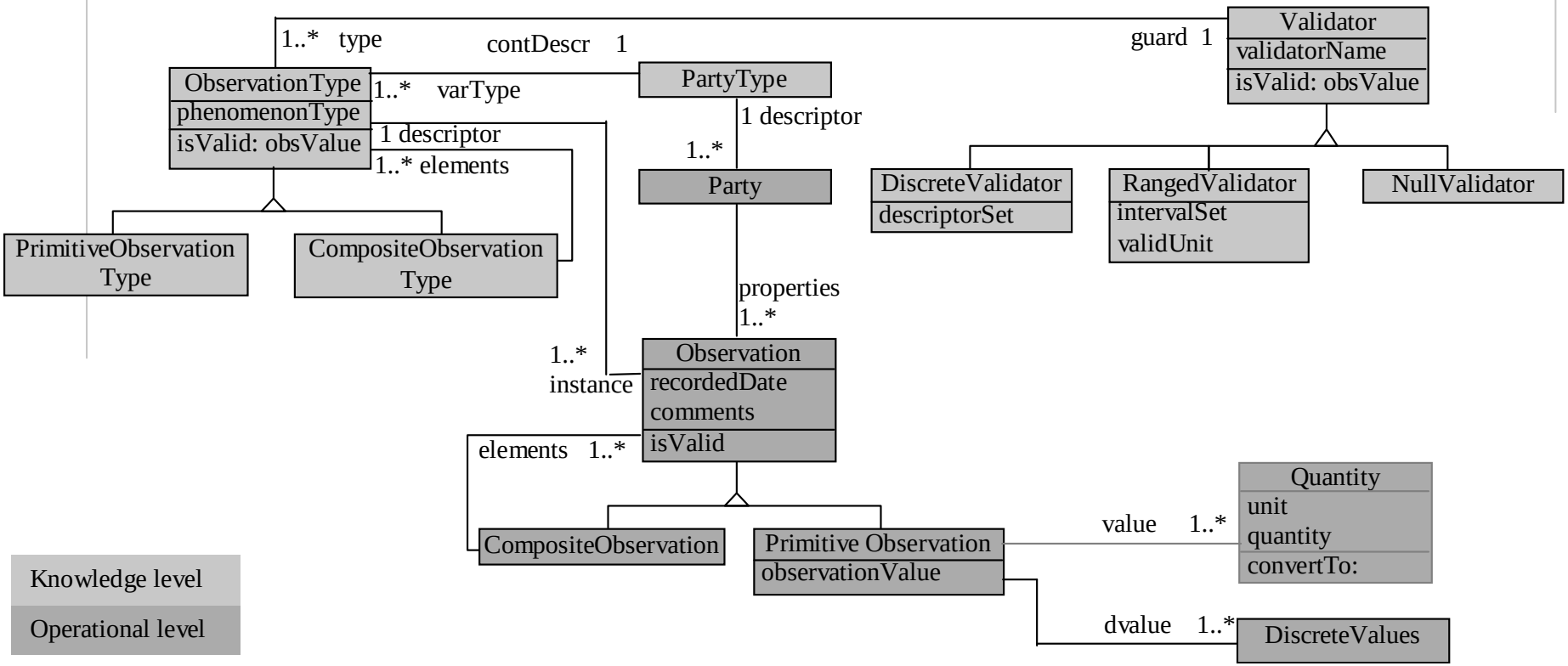
Framework's Design
(instance diagram)



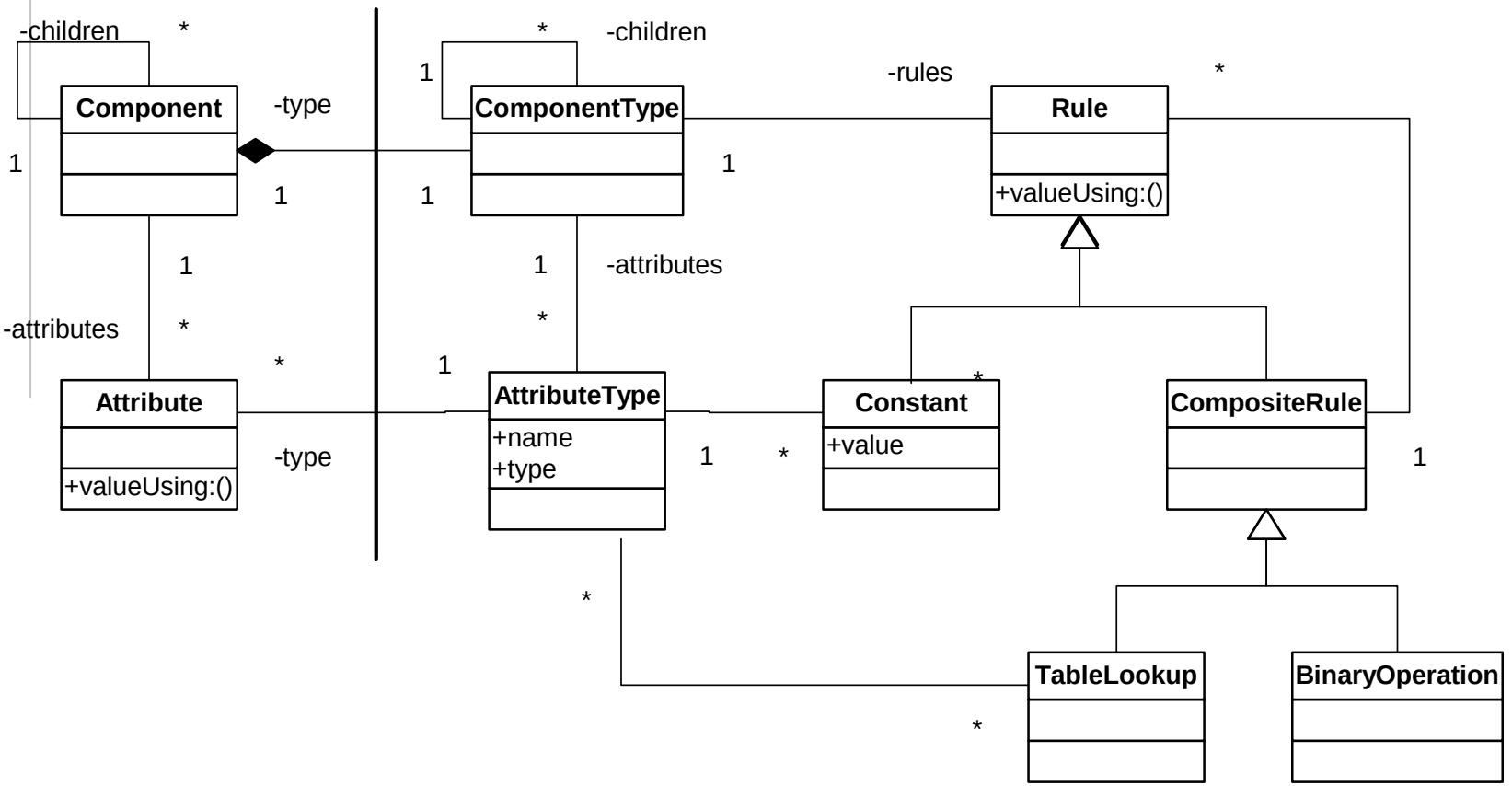
Framework's Design
(instance diagram)



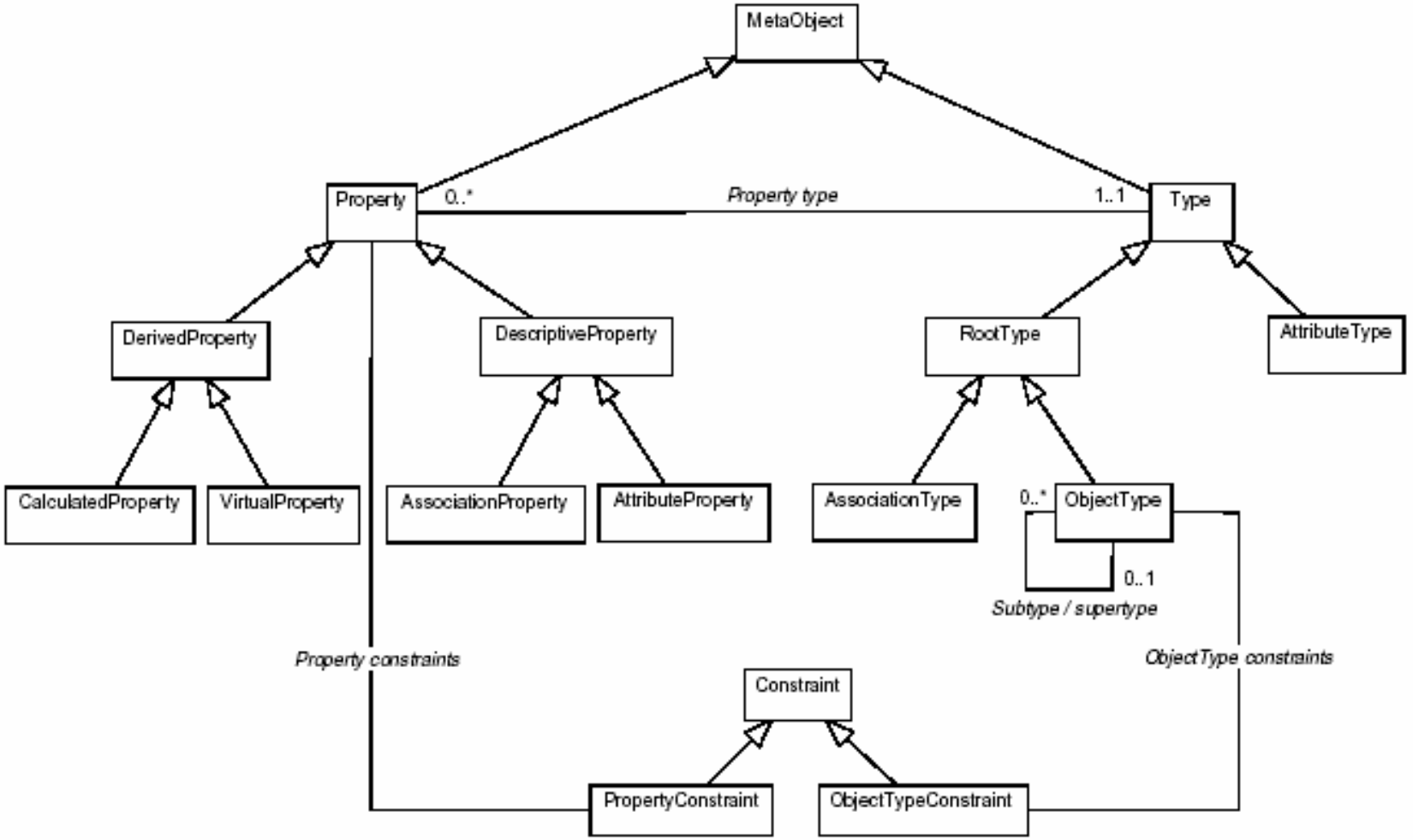
Medical Observation Example



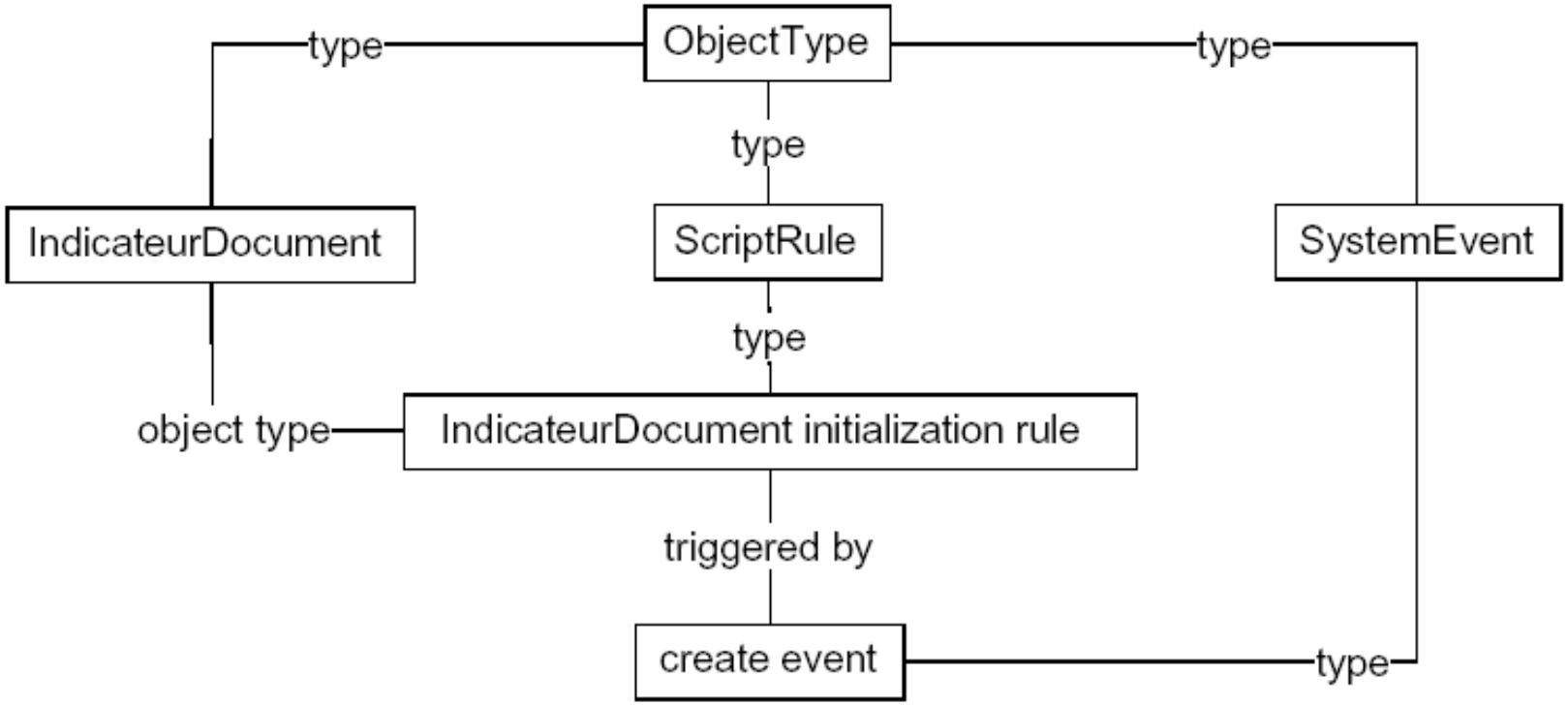
User Defined Product – Example "Hartford Insurance Policies"



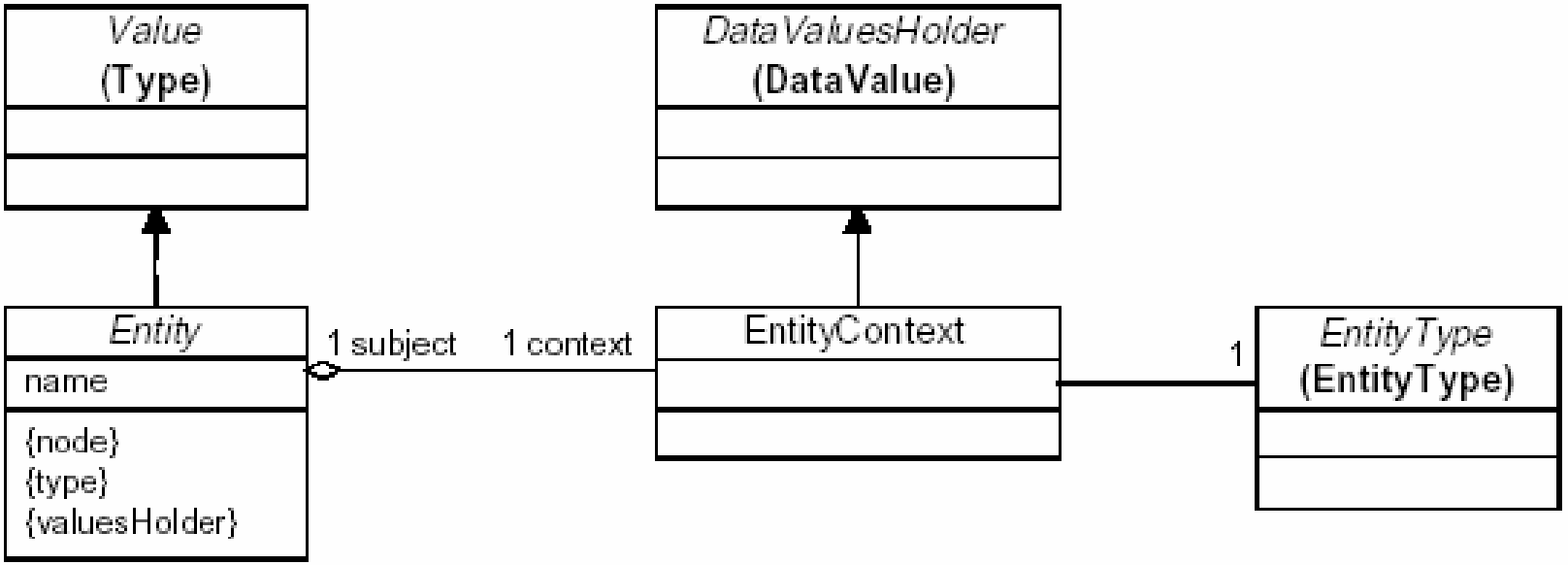
Argos Meta-Architecture (document workflow example)



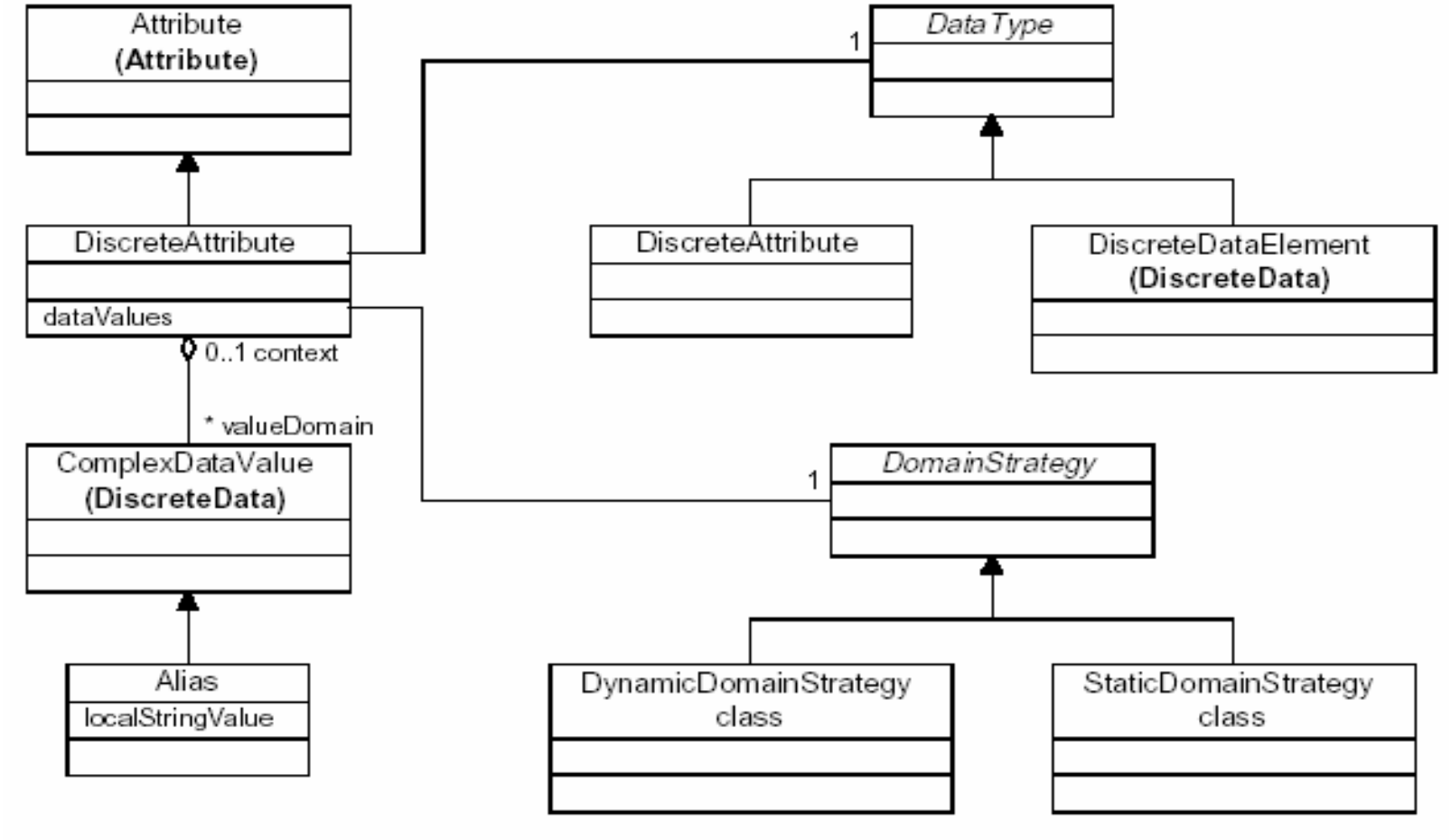
Argos Business Rule
(document workflow example)



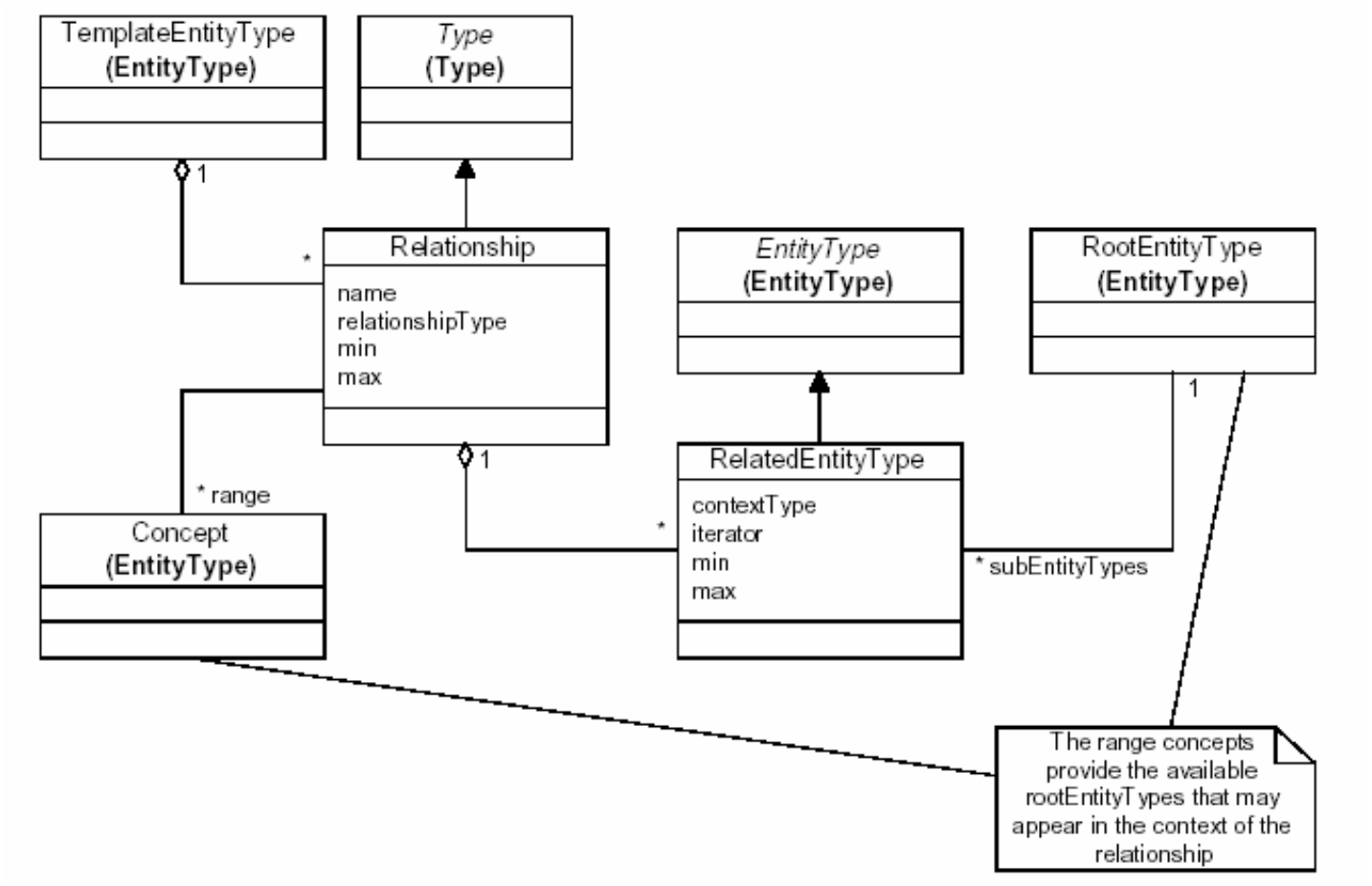
Objectiva Business Entities
(telephony example)



Objectiva Attributes
(telephony example)



Objectiva Entity Relationships (telephony example)



Precautions

Avoid using the metadata for storing:

- Error and warning messages to the user.
- Relationships between classes of the model (example: ObservationType-Validator)
- Variables that are inherent to the design (example: RangeValidator::unit)
- Over design…

Other Issues

- Consistency (versions)
- Dynamic GUIs
- Managing Releases
- Editors
- Optimizers

Maintaining consistency (versions)

- It is important to maintain consistency between the metamodel and any changed instances of TypeObject or other object associated with them.
  - Example: changing the legal range of a Validator can make existing observations invalid.
- May have to keep version of the metadata available and apply the rules based upon the timeframe the rule applies.

Metamodel and GUI

Metamodel and GUI

- The metadata can simplify building user interfaces. Special GUI components can be developed for using the metadata.
- Example: The Observation model includes widgets that display list of values from the DiscreteValidators and also EntryBoxes that use RangeValidator.
- A Mediator and Adaptor layer was developed for managing the interactions between the domain objects and the GUIs.

Metamodel and GUI

- Generating Dynamic GUIs is Hard!
- Can generate GUIs using metadata.
- Special GUI components can be developed for using the metadata.

Managing releases

- The system has releases because of changes in the metadata not only the code.
- Changes in the metadata should be checked by running test cases. Use of testing tools is recommended.
- Versions of the metadata has to be kept.
- May have effective dates for the rules which are represented by the metadata along with other related history patters.

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 (OMG)

Black-box frameworks

- These frameworks are instantiated by means of parameterization and object creation.
- They don't need to have a meta-level.
- They don't need to have interpreters and builders.
- They both use very similar patterns (type-objects, properties, strategies, …)

AOMs can be Black-box frameworks
but don't have to be--and vice-versa

Code Generators

- It provides infrastructure for transforming descriptions of a system into code.
- Descriptions are based on provided primitive structures or elements.
- Code generators produce either executable-code or source-code.
- Can use metadata and editors for describing code to generate.

Metamodeling techniques

- It focus on manipulating the model and meta-model behind a system as well as supporting valid transformations between different model representations.
- The attention is on the meta-model, or a model or generating a model, rather than the final application that will reflect the business requirements.
- This technique is used for describing the domain-specific language.

Domain Specific Languages

- DSLs can be a scripting language and work by means of parameterization.
- They don't need to have a meta-level but they often do.
- They don't need to have interpreters and builders.
- They both use solving similar patterns, they just might do it in different ways.

AOMs can be a Domain Specific Language
but don't have to be--and vice-versa

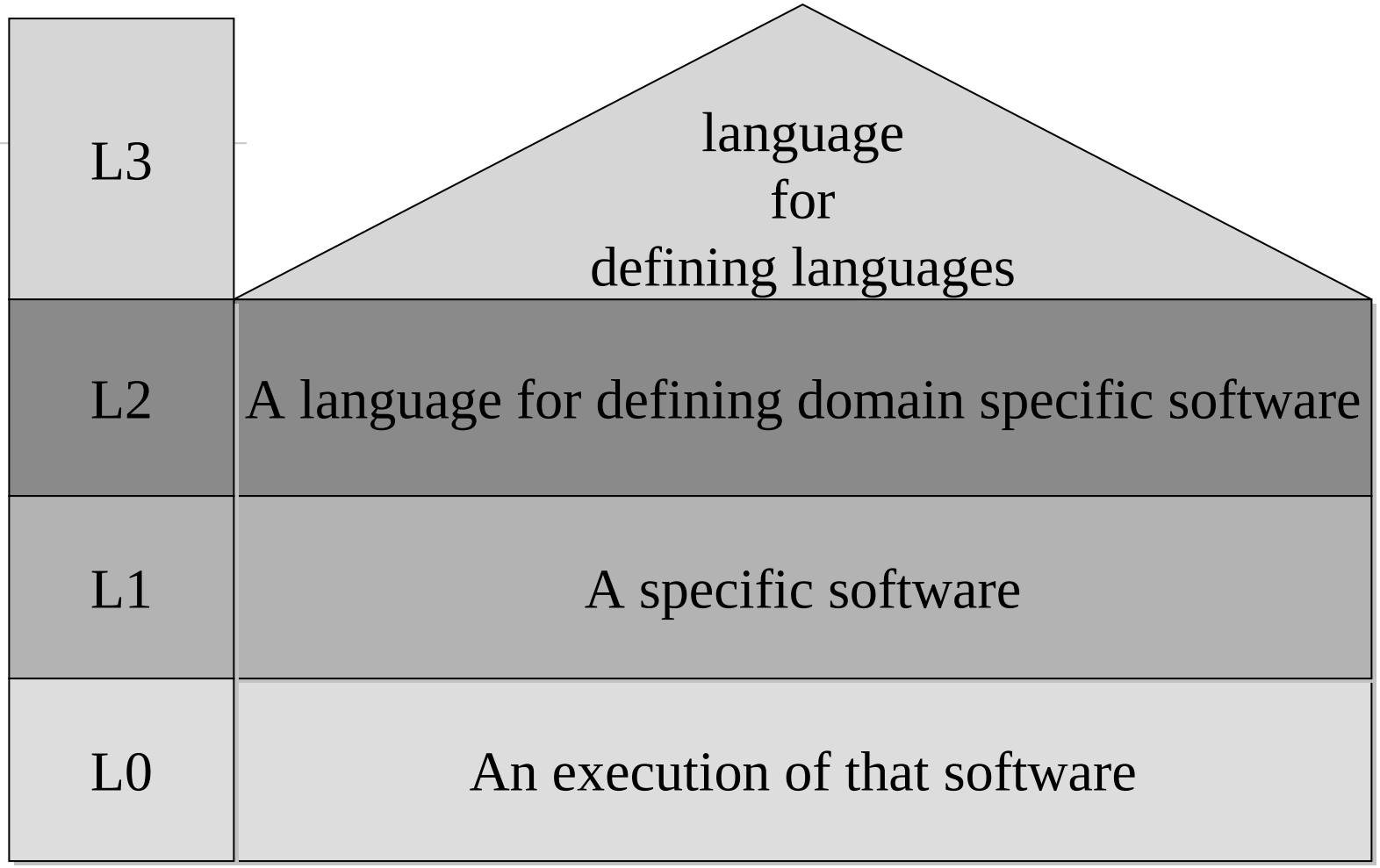
Table-driven Systems

- Business rules can be parameterized and stored in a database.
- The running system can either interpret these rules from a database table or the appropriate function can be called with the differing values from the database.
- Sometimes these are implemented with triggers and stored procedures.

Model Driven Architecture

- MDA is highly related to AOMs
- UML Virtual Machine is an AOM
- UML Virtual Machine is an MDA approach
- Focuses more on the modeling perspective

Dimensions of Abstraction



Dimensions of abstraction in Adaptive Object-Models, Reflection and OMG ’s metamodeling Architecture

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

Advantages of Adaptive Object-Models

- Can more easily adapt to new business requirements.
- Smaller in terms of classes so possibly easier to maintain by experts.
- Changes do not require recompiling the system.
- Business People can make changes.
- Time to market can be reduced.

Disadvantages of Adaptive Object-Models

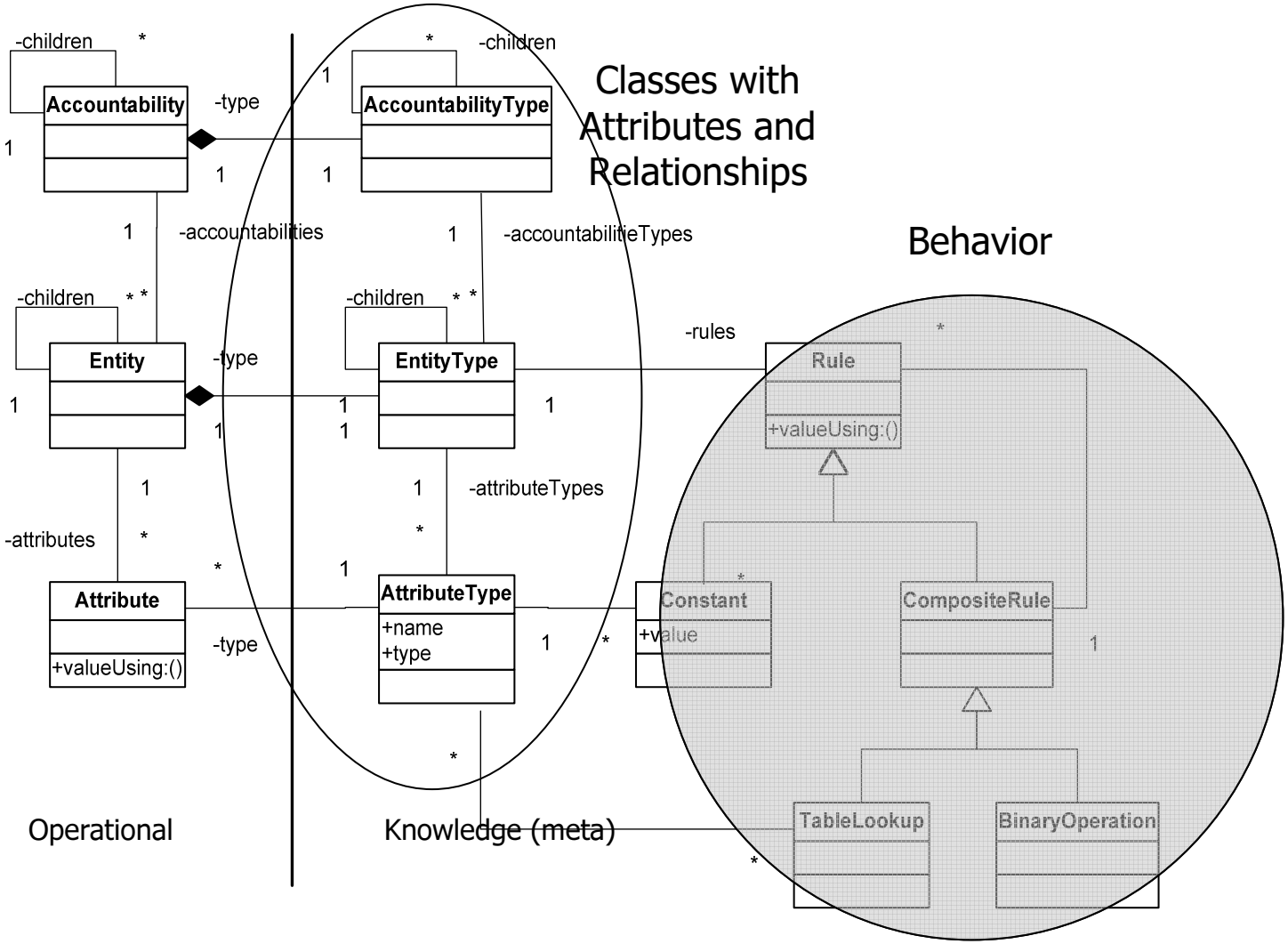
- It demands having infrastructure for storing, building, interpreting metadata.
- Developing AOM can be expensive.
- Can be hard to understand and maintain.
- It requires skilled human resources.
- Can have poor performance.

Process for Developing AOMs

- Developed Iteratively and Incrementally.
- Get Customer Feedback early and often.
- Add flexibility only when and where needed.
- Provide Test Cases and Suites for both the Object-Model and the Meta-Model.
- Develop Support Tools and Editors for manipulating the metadata.

Very similar to Evolving Frameworks by Roberts and Johnson PLoPD3

Adaptive Object Model "Common Architecture"



Summary

- Adaptive Object-Models can take time to develop -- but the payoff can be enormous!
- Adaptive Object-Models work based upon domain expert knowledge.
- Adaptive Object-Model architectural style exposes the elements of the domain and business rules.
- Applying well-known design principles (e.g. TypeObject, Properties, Entity-Relationship, and Strategies/RuleObjects) works well for developing systems that can dynamically adapt to your changing business environment.

Summary

- AOMs Separates what changes quickly from what changes slowly (hot-spots).
- Takes into account who changes what, when, and where.
- AOM Objects constitute a domain specific language.
- Building languages out of objects can be good…reflection guys say this!

Where to Find More Information

- http://www.adaptiveobjectmodel.com
- http://st-www.cs.uiuc.edu/users/droberts/evolve.html
- http://www.joeyoder.com/papers/patterns
- http://hillside.net
- http://st-www.cs.uiuc.edu/
- http://www.refactory.com

That's All