Ultimate Agility: Let Your Users Do Your Work!

Joseph W. Yoder

joe@refactory.com

http://www.refactory.com

Agenda

- Agility and architecture to support change
- Adaptive Object-Model (AOM) basics
- An example: Unfolding the core architecture
- An AOM Pattern Language:
- Case studies and conclusions

Evolved with Rebecca Wirfs-Brock

Slide - 2

Motivation: Need to Quickly Adapt to Change

Business Rules or Domain Elements are changing quickly:

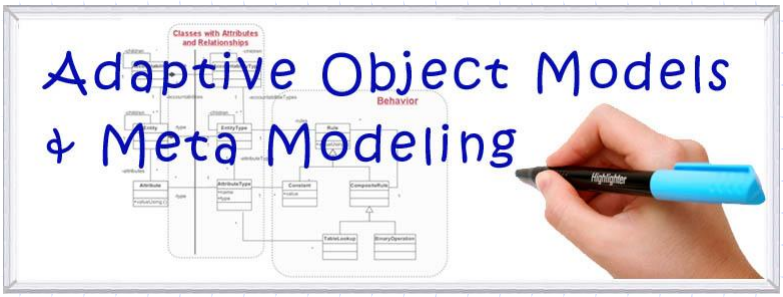
- New calculations for insurance policies and new types of policies offered
- Online store catalog with new products and services and rules applying to them
- New cell phone product and services…

Need a quick way to **develop** and **adapt** to these **changing requirements**. Typical way requires **iterations**, **build-compile-release**, lots of **testing** and **deployment** through **change control**!

Slide - 3

Motivation: Design for easy change

Sometimes it pays to build systems so they can be changed or extended without hand coding all changes.



Slide - 4

Contrasting: Two Ways to Change a Working System

	Traditional SW Architecture	Adaptive Object-Model Way
Who implements the change?	A programmer after understanding and implementing a "story" card.	A domain expert does, if it is a domain object or business rule change.
How are changes verified?	Programmer writes tests, QA writes and runs acceptance tests, end-user approves.	Still have to run all unit and acceptance tests. But can also build into end-user tool checks that model changes won't "break" the existing working system.
How often can the system be updated in production?	At the end of an iteration.	Whenever changes are verified. Not tied to dev cycle.
What's the big deal?	Still have to go through some release cycle. Programming and deployment can become bottlenecks.	Significant changes can be made by end-users. Releasing can be as simple as updating production metadata.

Adaptive Object-Model (Active|Dynamic Object-Model)

- An ADAPTIVE OBJECT-MODEL provides meta information about the domain model and business rules that is interpreted.
- Typically arise from domain-specific frameworks refactored when they become too complex.
- Allow systems to quickly adapt to changing requirements.

Slide - 6

Adaptive Object-Model Basics

- 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.

Slide - 7

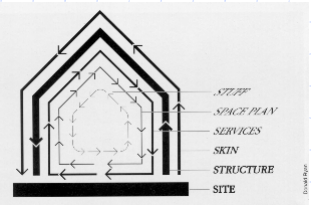
Adaptive Object-Model General Design Principles

- Identify what is changing at a rapid pace.
- Separate and isolate changing parts.
- Build rapidly changing domain objects or behaviors so they can be evolved without recompiling code.

Slide - 8

Stuart Brand's Shearing Layers

- Buildings are a set of components that evolve in different timescales.
- Layers: site, structure, skin, services, space plan, stuff. Each layer has its own value, and speed of change (pace).
- Buildings adapt because faster layers (services) are not obstructed by slower ones (structure).



—Stuart Brand, *How Buildings Learn*

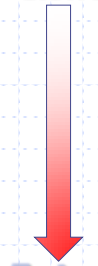
Slide - 9

Yoder and Foote's Software Shearing Layers

- "Factor your system so that artifacts that change at similar rates are together."—Foote & Yoder, Ball of Mud, PLoPD4

Layers

Slower



Faster

- The platform
- Infrastructure
- Data schema
- Standard frameworks and components
- Abstract classes and interfaces
- Classes
- Code
- Data

Slide - 10

Agile Values

- Core values:
  - Design Simplicity
  - Communication
  - Teamwork
  - Trust
  - Satisfying stakeholder needs
- Constant learning

Slide - 11

Can you be agile if you create an AOM?

- **Yes, because it enables change.**
- But you must justify extra complexity.
- And counteract arguments:
  - How can anything that involves frameworks or metadata ever be considered agile?
  - An AOM isn't simple design. And all we should do is simple design.

Slide - 12

Agile Myths

- The goal is always simple design. So…
  - never create frameworks.
  - never write code that isn't immediately self-explanatory.
  - if I don't understand your code, your design is too complex.
- Simple solutions are always best.
- Building in flexibility is always over-engineering.
- You can change the system fast!!!

Slide - 13

Our Agile Design Values

- Respect your system's shearing layers.
  - Understand the rates of what changes.
- Determine who should be able to make changes, when, and at what cost.
  - Support them.
- Make what is too difficult, time consuming, or tedious easier.
  - Create tools, leverage design patterns, use data to drive behavior…
- Don't overdesign!!!

Slide - 14

Allowing Systems to Adapt

- Requirements change.
- Business Rules often change rapidly.
- Understanding of the domain changes, too.
- Domain Experts know their domain best.
- Applications have to quickly adapt to new business requirements.
- Agile development embraces changes to the requirements, even late in the development!
- One way to adapt is to refactor to support new business requirements…

Slide - 15

Elements of Adaptive Object Models

- Metadata
- TypeObject
- Properties
- Type Square
- 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.

Slide - 16

The Power of Metadata

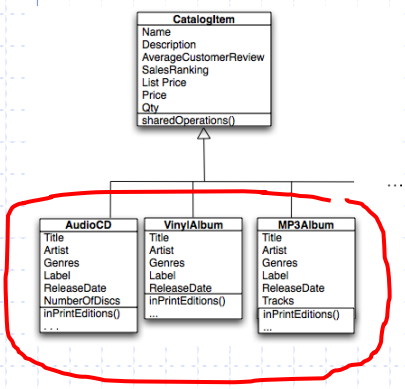
- Code is data, data is code. Everything is data. And data can drive behavior.
- Meta data simply describes other data.
  - "If something is going to vary in a predictable way, store the description of the variation in a database so that it is easy to change"—Ralph Johnson.

Slide - 17

Type-Object

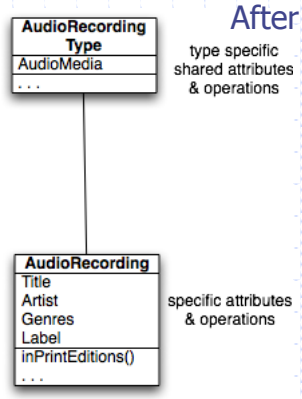
PLoPD3 - Johnson and Woolf

Before



Symptom: Explosion of classes based on minor attribute differences

After

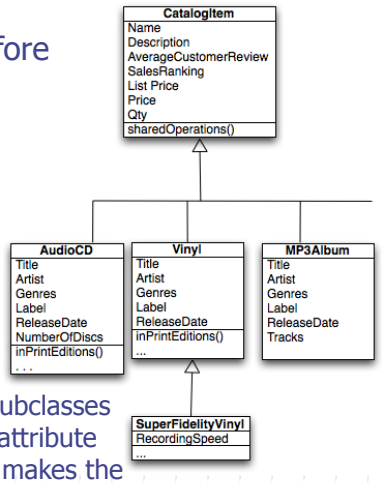


Solution: Factor common attributes into "type" classes

Slide - 18

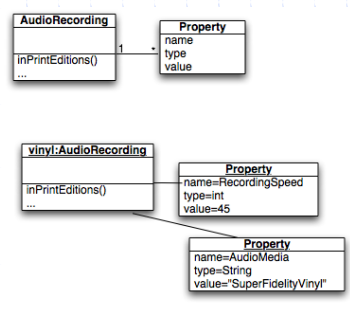
Properties

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.

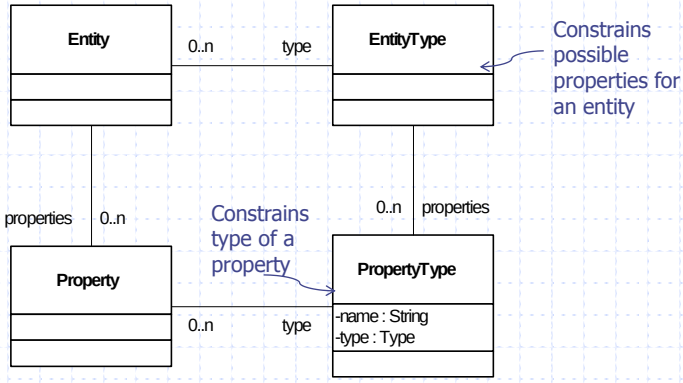
Slide - 19

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?

Slide - 20

TypeSquare

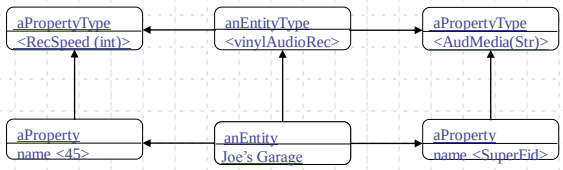


Example: Now it is easy to add different kinds of catalog items

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

Slide - 21

Type Square
(instance diagram)



Slide - 22

TypeSquare Exercise

- GROUP EXERCISE…

Slide - 23

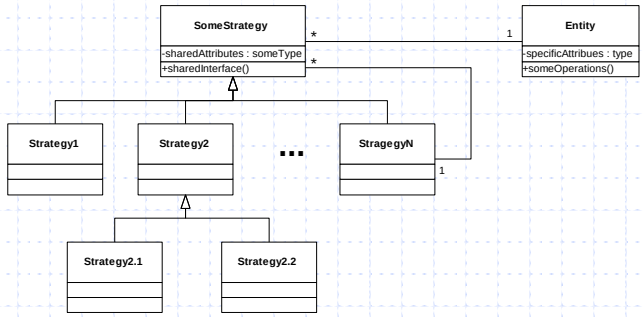
Dealing with Behavior/Rules

- 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.*

Slide - 24

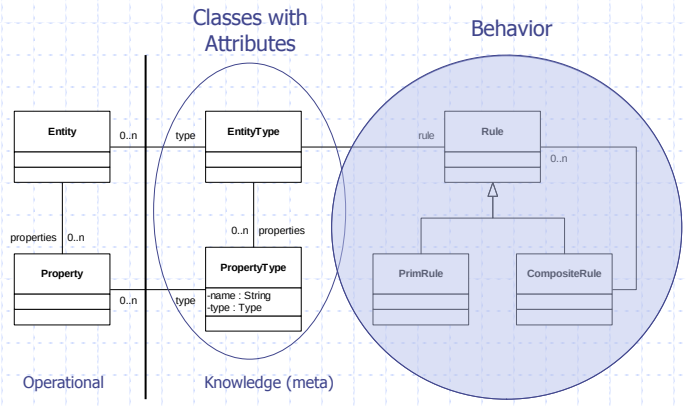
Strategies/RuleObjects Solution
(Behavior/Methods)



Design Patterns - GOF95

Slide - 25

Putting It All Together
(Very Common Structure)



ECOOP & OOPSLA 2001 Yoder, Balaguer, Johnson

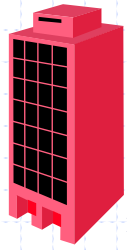
Slide - 26

An AOM Example...Refactoring as We Go

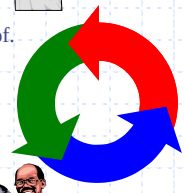
Slide - 27

Newborn Screening Refactoring Example

Doctor, HealthProf.



Hospital, Lab

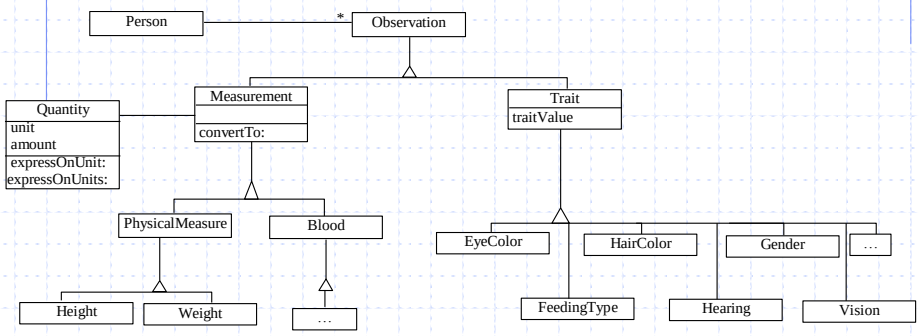


Mother, Infant

Blood Specimen

Slide - 28

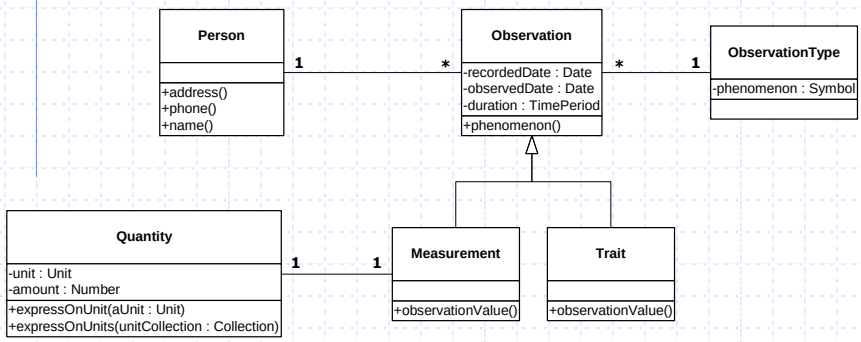
Medical Observation – Basic OO Design Model



What happens when a new observation is required?

Slide - 29

Observation Design (1st Design)

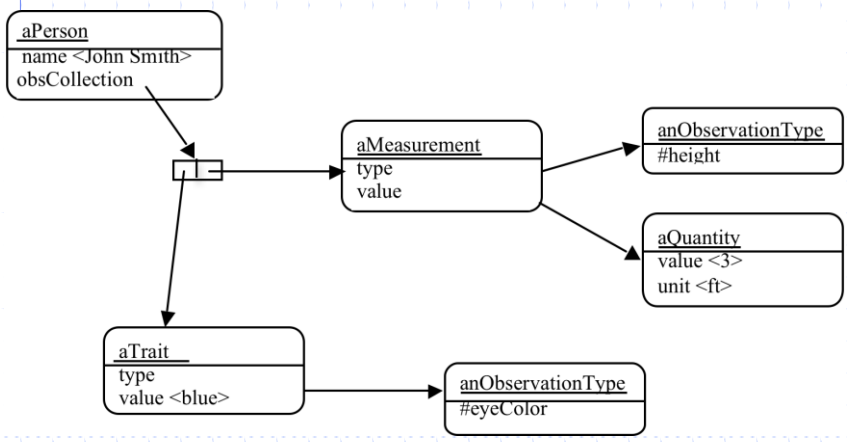


Slide - 30

Observation Design
Example

Slide - 31

Observation Design
(instance diagram)



Slide - 32

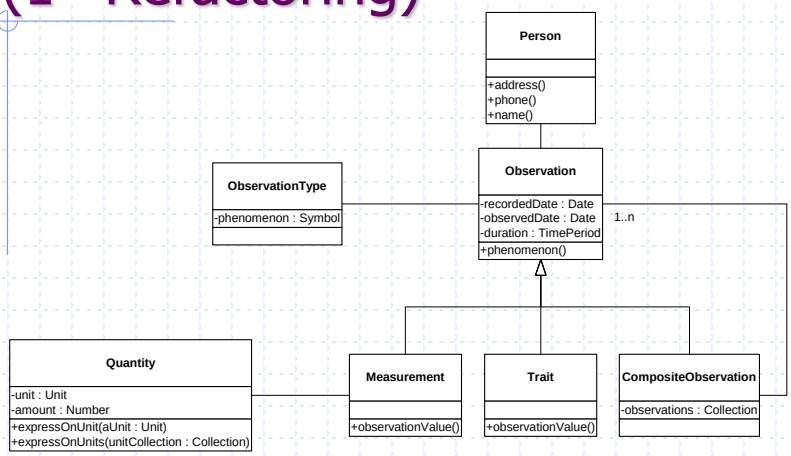
Composing Observations

Observations can be more complex

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

Slide - 33

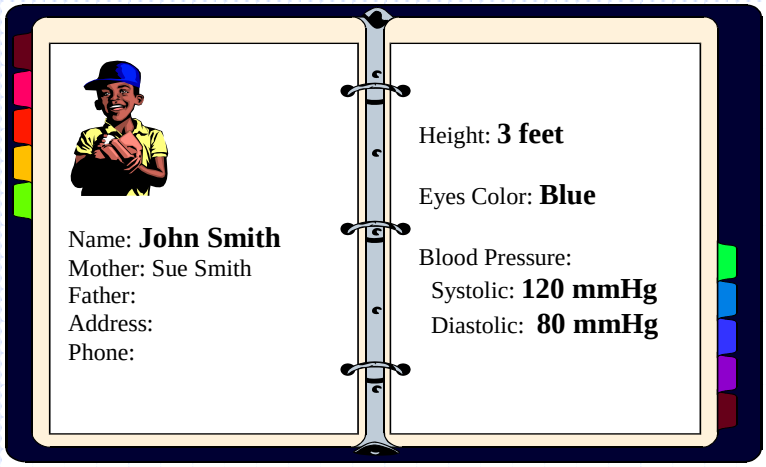
Composite Observation Design (1st Refactoring)



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

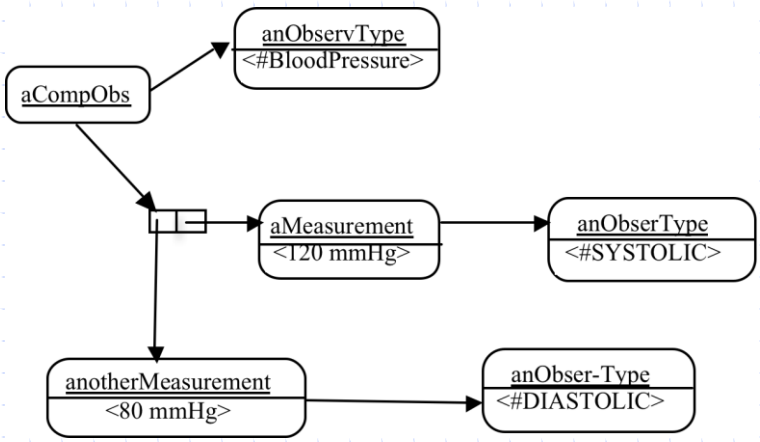
Slide - 34

Observation Design
Example



Slide - 35

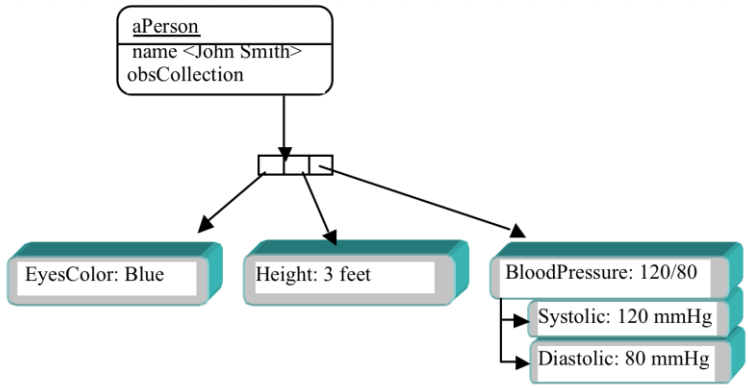
Composite Observation Design
(instance diagram)



Slide - 36

Composite and Primitive Observation Design

What we know about John?



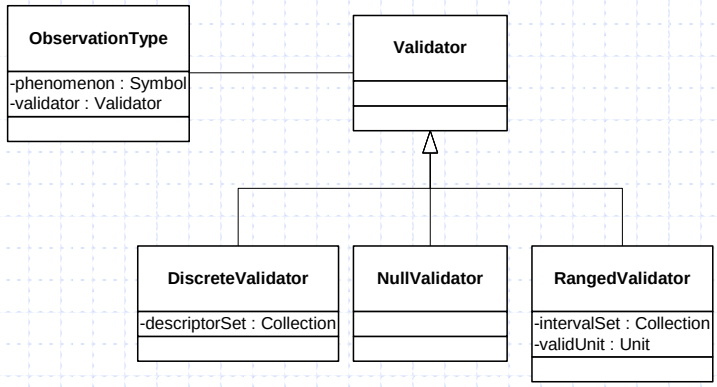
Slide - 37

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

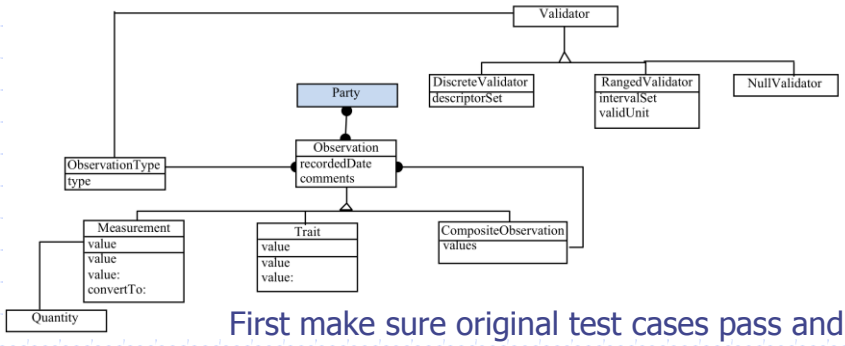
Slide - 38

Validating Observations Design (2nd Refactoring)



Slide - 39

Overall Observation Design



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

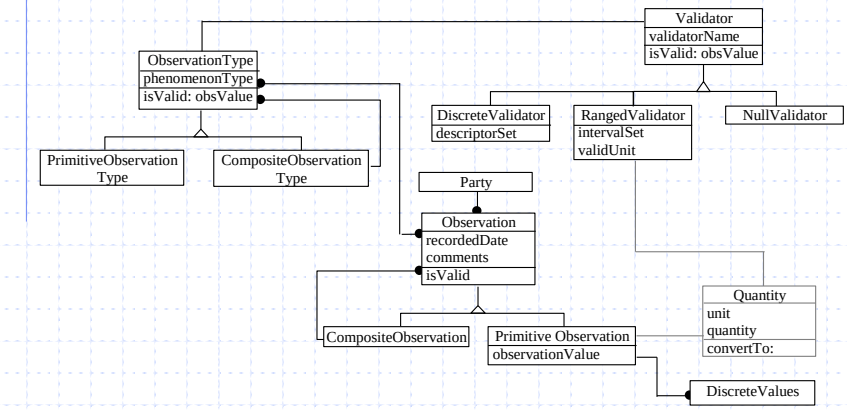
Is everything an Observation?

How does the model specify the structure of the Composite?

What is the relationship between Trait and DiscreteValidator?

Slide - 40

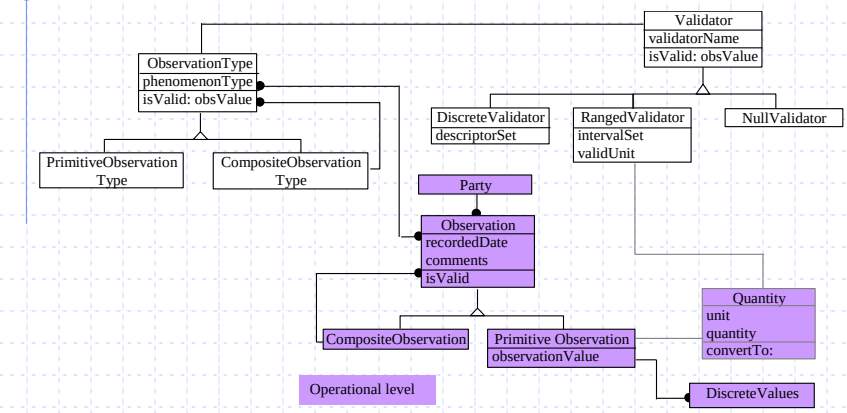
Observation Design



Extend the design by adding Composite to Type
Refactor the Metadata!

Slide - 41

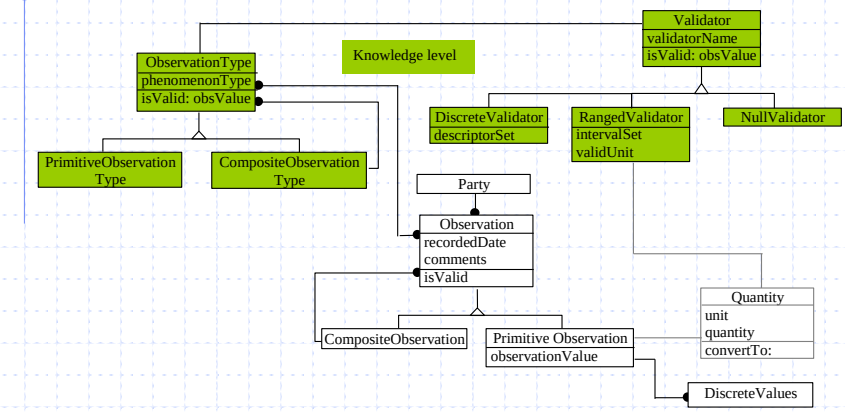
Observation Design



Extend the Design by adding Composite to Type
Refactor the Metadata!

Slide - 42

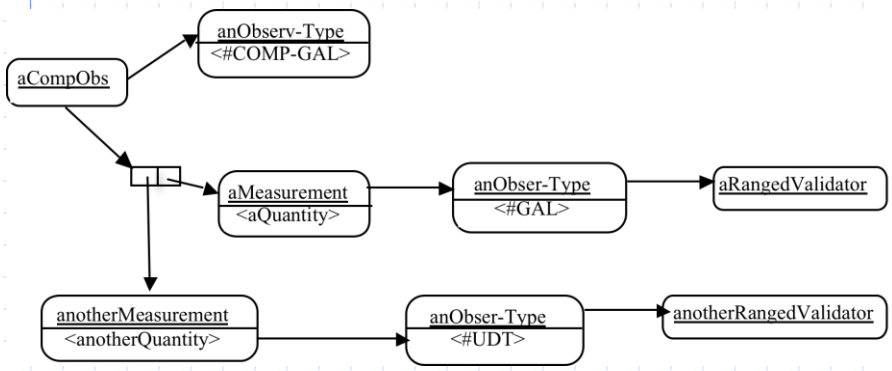
Observation Design



Extend the design by adding Composite to Type
Refactor the Metadata

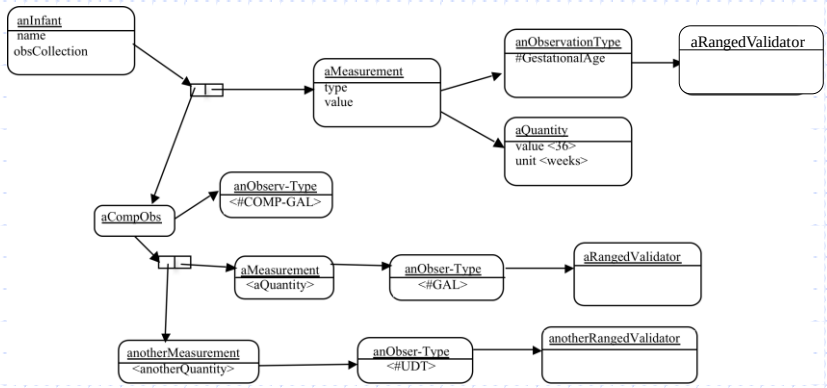
Slide - 43

Observation Design
(instance diagram)



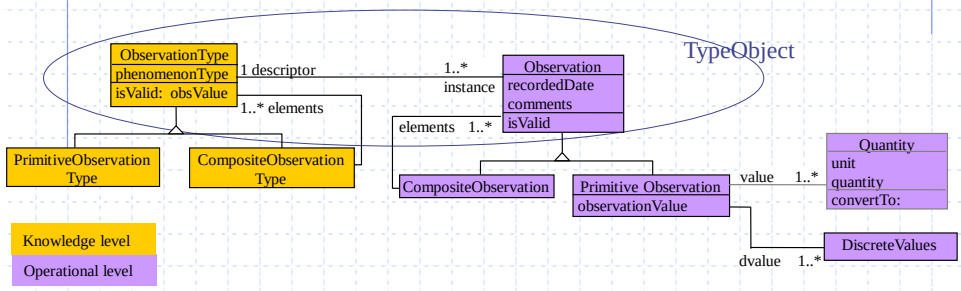
Slide - 44

Observation Design
(instance diagram)



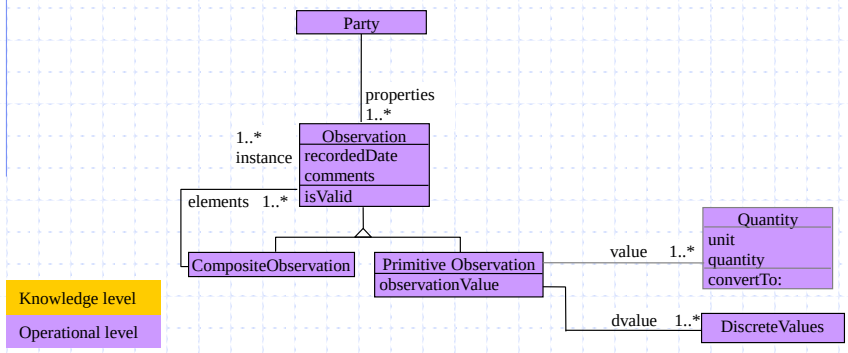
Slide - 45

Observations: TypeObject



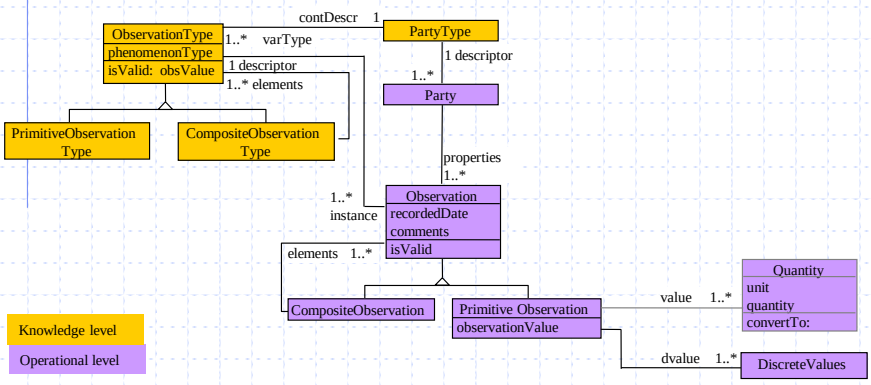
Slide - 46

Observations: Properties



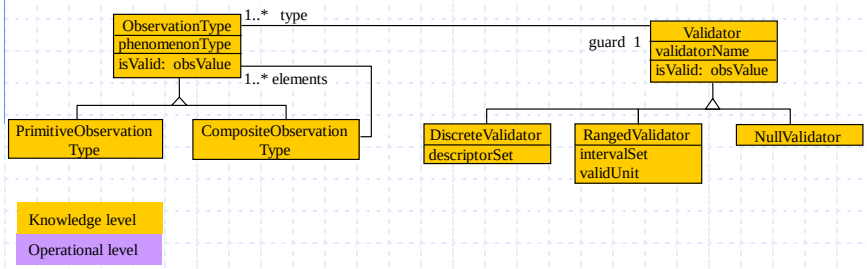
Slide - 47

Observations: TypeSquare



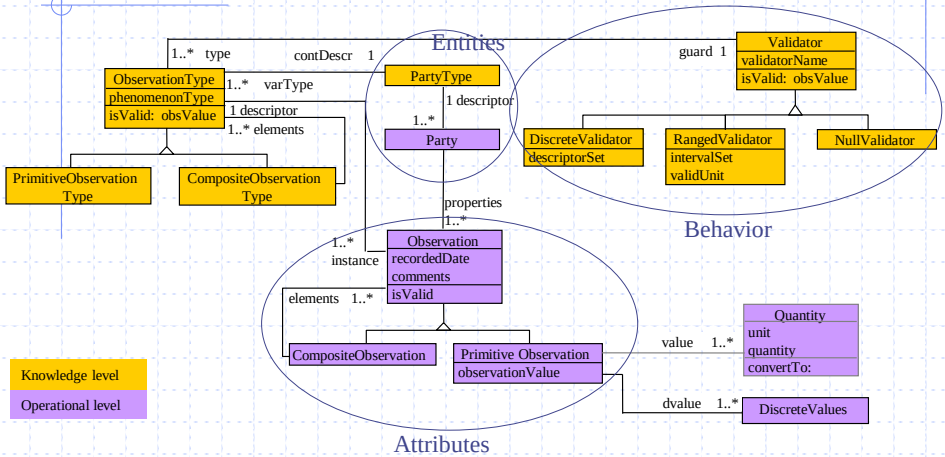
Slide - 48

Observations: Strategy



Slide - 49

Medical Observations Design



Slide - 50

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.

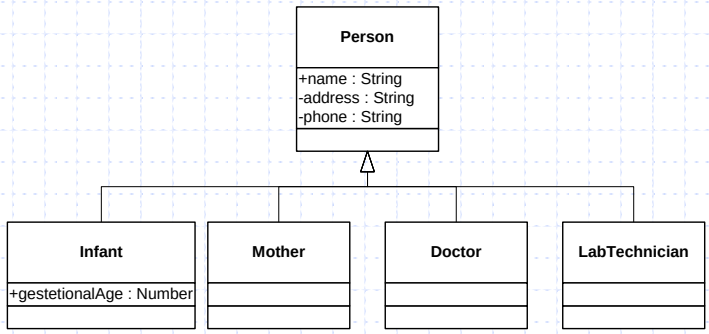
Slide - 51

Extending our Example to Include...

Slide - 52

Entities and Relationships

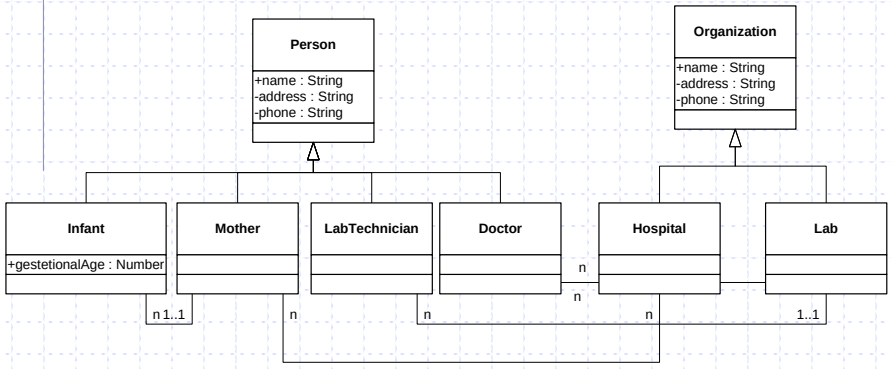
Infants, Mothers and Doctors...



Slide - 53

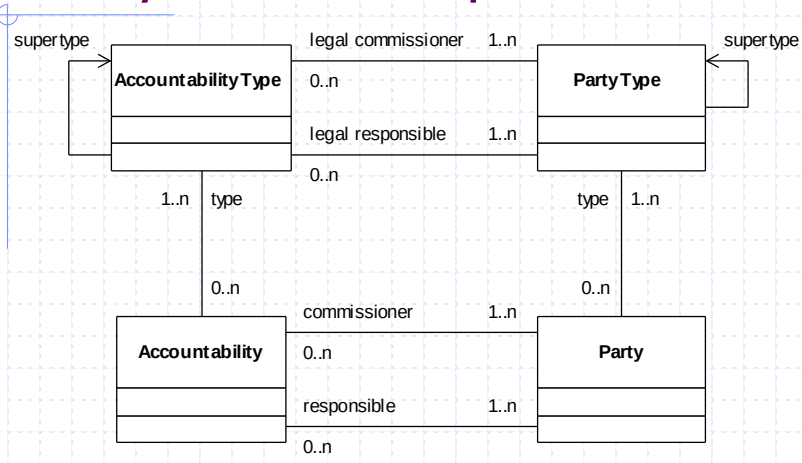
Newborn Screening

Putting it all together



Slide - 54

Entity-Relationship Patterns



Analysis Patterns – Martin Fowler

Slide - 55

Party and Accountability

Modeling relationships between entities

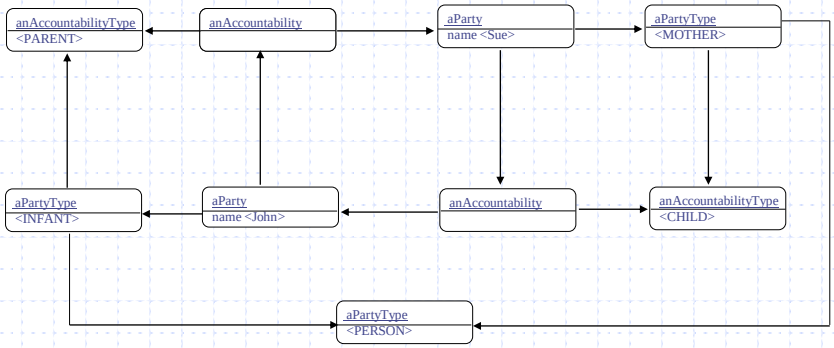
Sue Smith

John Smith

Sue is the mother of John

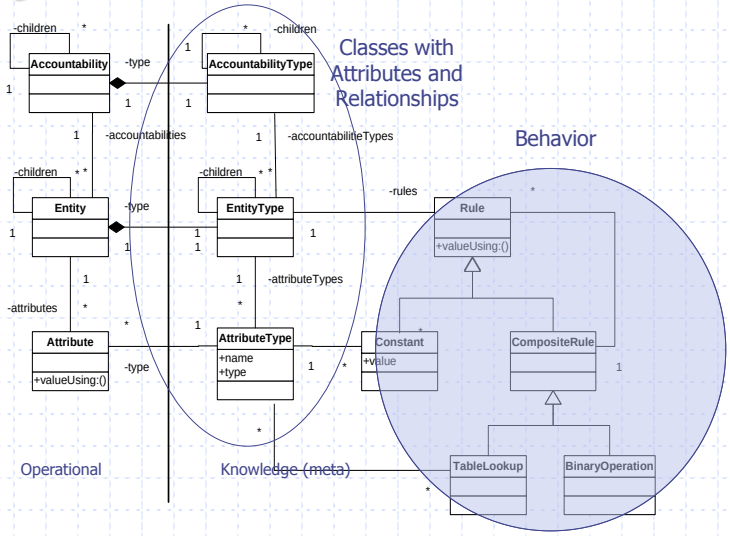
Slide - 56

Party and Accountability
(instance diagram)



Slide - 57

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



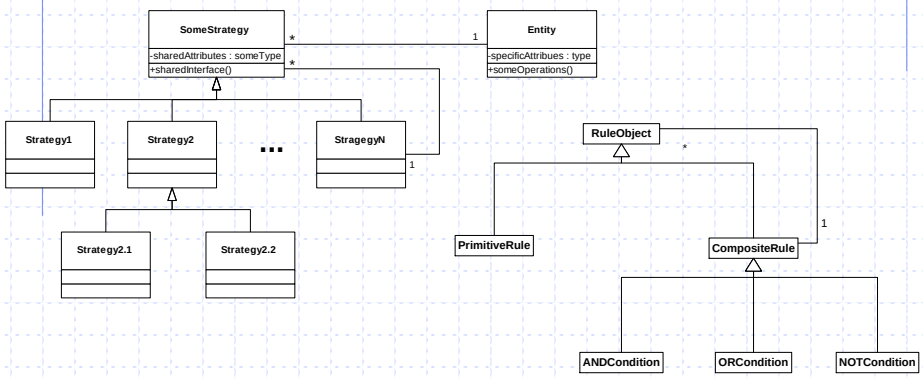
Slide - 58

Different Types of Behavior

- Behavior is very domain specific:
  - Constraints on values, relationships, state changes…(we've looked at some of these)
  - Functional…(invoicing, insurance policies,…)
  - Workflow…(docs through a school, PoD,…)
  - Event based …(cancel order, exception,…)
- Maybe can organize differently but most Rules/Behaviors can be described above

Slide - 59

Strategies/Interpreters/RuleObjects (Behavior/Methods)



Design Patterns - GOF95

Composite Strategies ➔ Interpreter

Slide - 60

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

Slide - 61

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.

Slide - 62

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, ...

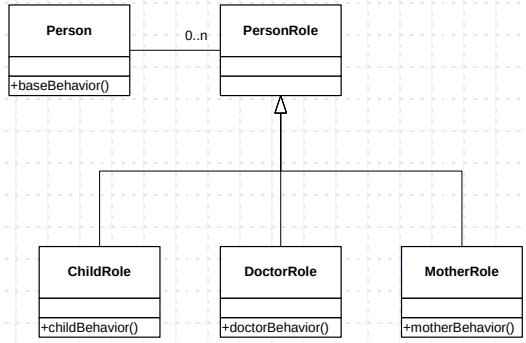
Slide - 63

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.

Slide - 64

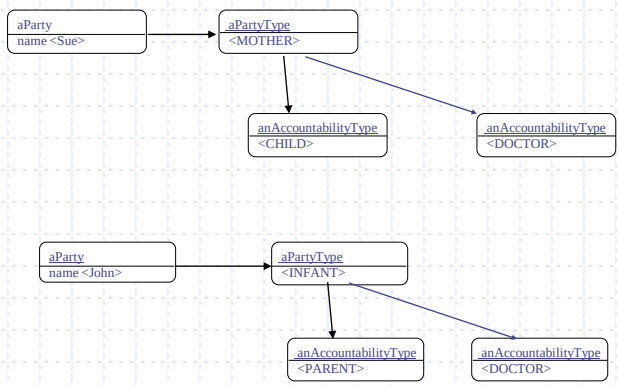
Roles
(an example)



PLoP 97 - Fowler
PLoPD4 - Baumer, Riehle, Siberski, Wulf

Slide - 65

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



Slide - 66

We have examined the "core" patterns for the domain model

What else is there?

How do you interact with the domain?

...

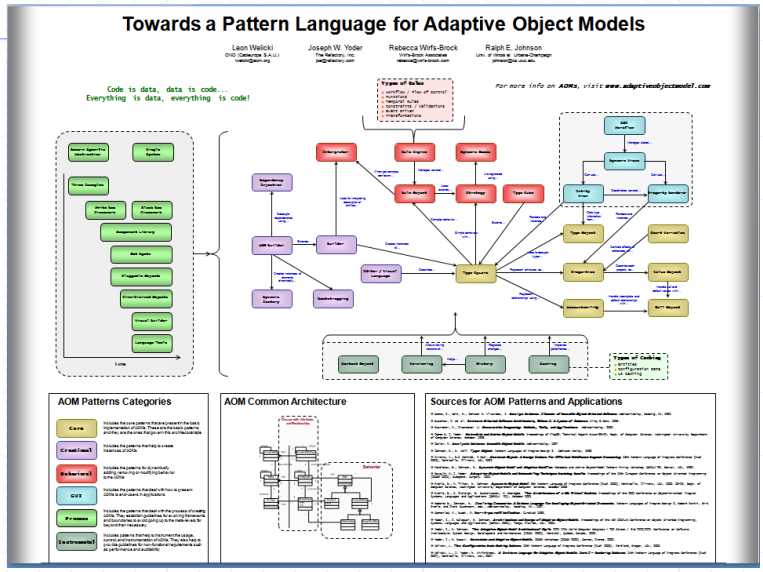
Slide - 67

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.

Slide - 68

OOPSLA 2007 Poster Session



Slide - 69

Other Issues

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

Slide - 70

Metamodeling techniques

- AOM is a specific kind of metamodeling technique that focuses on describing a domain model that will be reflected in a running application.
- Other metamodeling approaches focus on a meta-model for generating yet another model that can be generated, interpreted, executed, or compiled.
- Both of these techniques are commonly used for describing a domain-specific language.

Slide - 71

Precautions

Avoid using the metadata for storing:

- Error and warning messages to the user.
- Relationships between classes of the model (example: ObservationType-Validator).
- Attributes/ Variables that are inherent to the design (example: RangeValidator-unit).
- Over design… everything doesn't have to be totally meta-described.

Slide - 72

Persisting an AOM

- The metadata is expressed with objects that can be mapped to relational as well as object-oriented databases.
- And there's increasing interest in defining metadata in XML/XMI.

Slide - 73

Keeping consistency (versions)

- Need to maintain consistency within the metamodel when changing instances of TypeObject or other objects 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.

"Adaptive Object-Model Evolution Patterns", SugarLoaf PLoP 2010, Atzmon Hen-Tov, Lena Nikolaev, Lior Schachter, Joseph Yoder, Rebecca Wirfs-Brock

Slide - 74

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

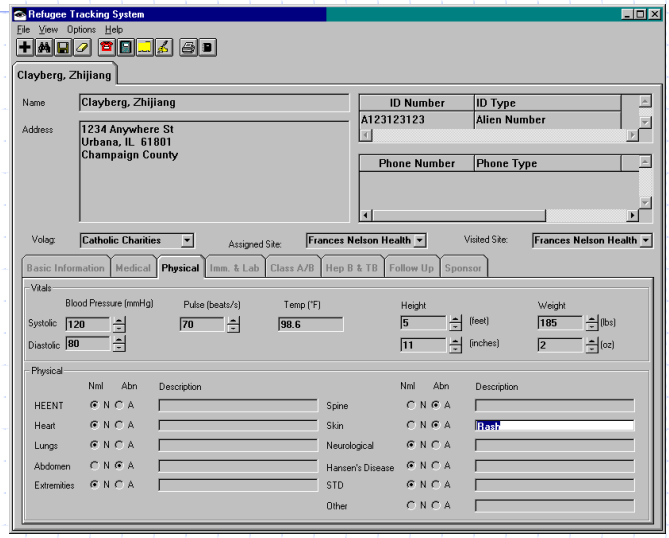
Slide - 75

Metamodel and GUI

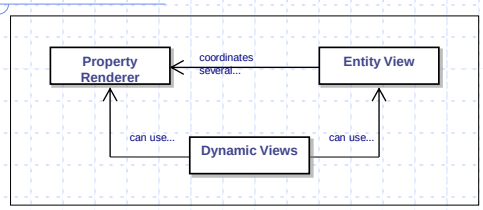
- The metadata can simplify building user interfaces. Special GUI components can be developed to use the metadata.
  - Example: The Observation model includes widgets that display list of values from the DiscreteValidators and also EntryBoxes that use RangeValidators.
- A Mediator and Adaptor layer was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

Slide - 76

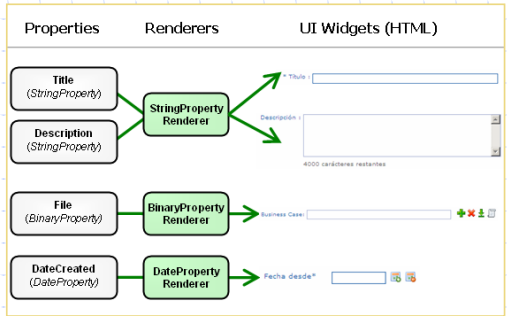
Metamodel and GUI



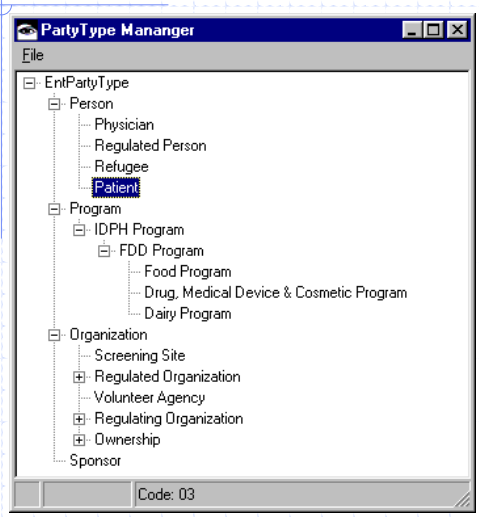
AOM Rendering Patterns

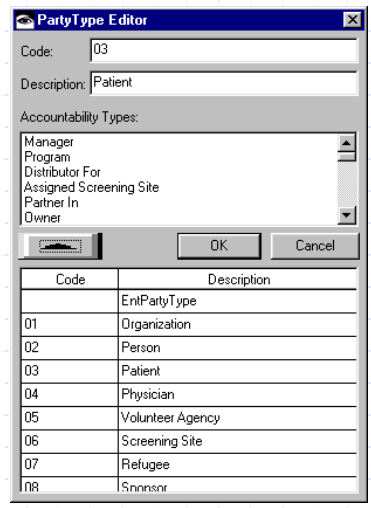


PLoP 2007 –
L. Welicki,
J.Yoder,
R. Wirfs-Brock



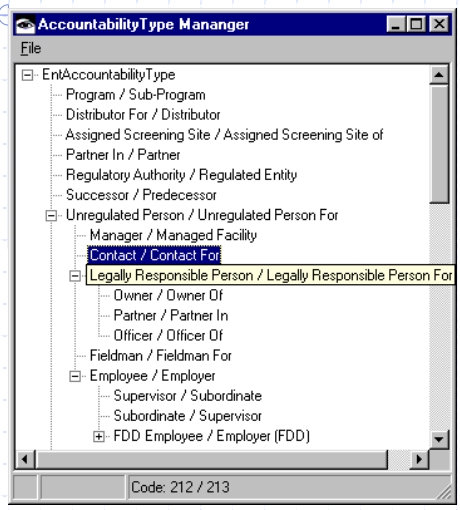
PartyType: Metadata-Editors

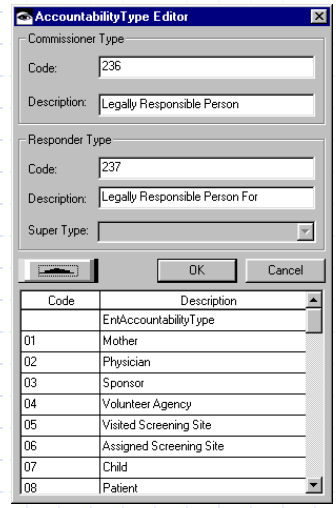




Slide - 79

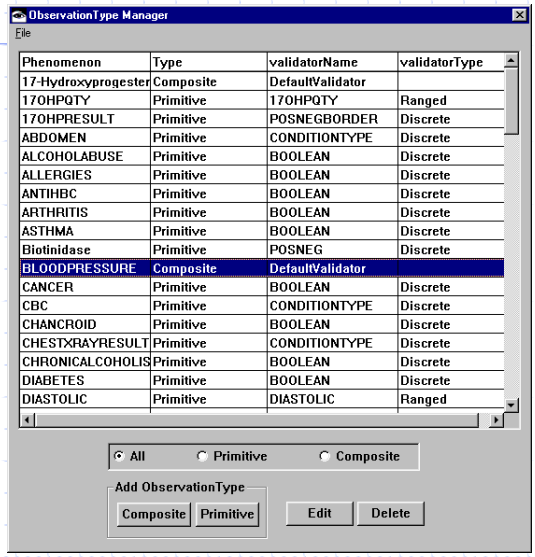
Accountability: Metadata-Editors



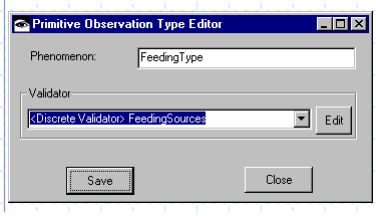


Slide - 80

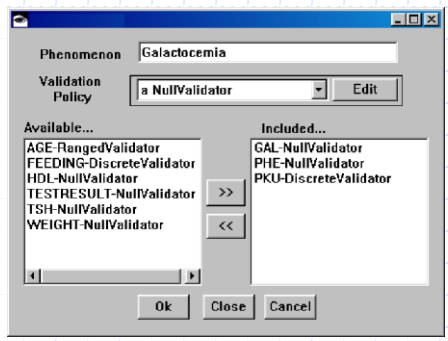
Observation: Metadata-Editors



Observation: Metadata-Editors

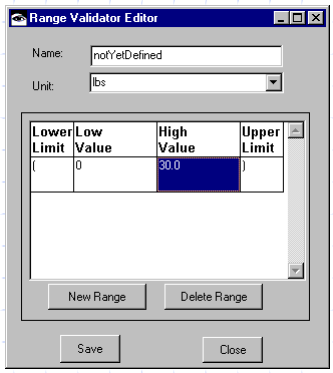


Primitive Observation Type Editor



Composite Observation Type Editor

Observation: Metadata-Editors



Slide - 83

Managing releases

- The system has releases because of changes in the metadata and the code.
- Changes in the metadata should be checked by running test cases.
- Multiple versions of the metadata may have to be supported.
- May have effective dates for the rules which are represented by the metadata.

"Adaptive Object-Model Metadata Evolver", PLoP 2010, Atzmon Hen-Tov, David H. Lorenz, Lena Nikolaev, Lior Schachter, Rebecca Wirfs-Brock, Joseph Yoder

Slide - 84

Interpreters / Builders: Problem

- Creating methods in each class for instantiating the required metadata defeats flexibility goals.
- 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.

Slide - 85

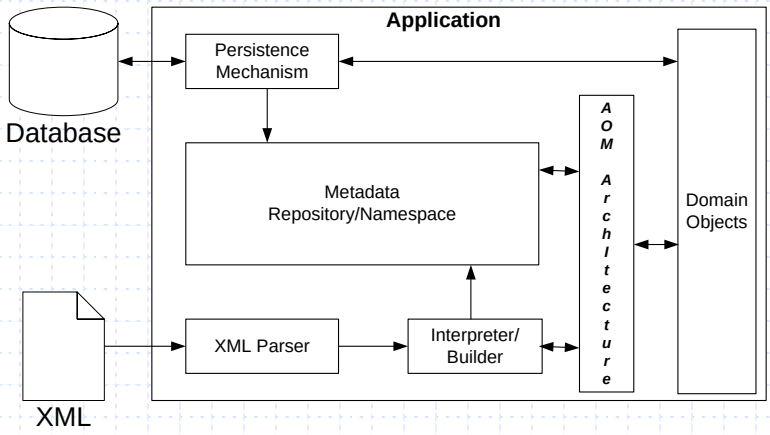
Interpreters / Builders: Context

- Adaptive Object-Models need to implement ways for describing the types of entities, properties, and relationships and create them from this description.
- They also need to interpret the dynamic behavior at runtime using strategies or rules.

"Adaptive Object Model Builder", PLoP 2009, *Leon Welicki, Rebecca Wirfs-Brock & Joseph W. Yoder*

Slide - 86

Interpreters / Builders "Virtual Machine"



Slide - 87

Other Examples

- Industrial (Manufacturing)
- Medical (IDPH)
- Insurance
- Telephony Billing System
- Pontis Telephony Marketing
- Portugal System

Slide - 88

Manufacturing - Shared Corporate Organization Data

- ~1,000,000 lines of COBOL code for editing organization data.
- Organization data shared by many systems.
- Had a common UI editor for adding, and updating this data. Editor was cloned and modified many times…grew to 100+ copies.
- Documentation and maintenance cumbersome.

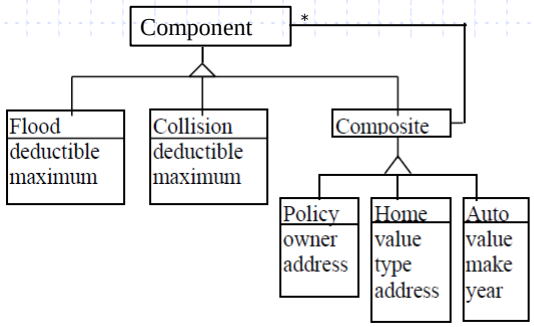
Slide - 89

Manufacturing - Shared Corporate Organization Data

- Rewrote in Java code ~ 40k LOC that used XML to describe the mappings to the organization data and generated the GUI and SQL queries.
- Documentation could be generated easily from XML mappings.
- Easy to change and release new versions for new tables through updates to the XML schema.

Slide - 90

User Defined Product
(insurance example)



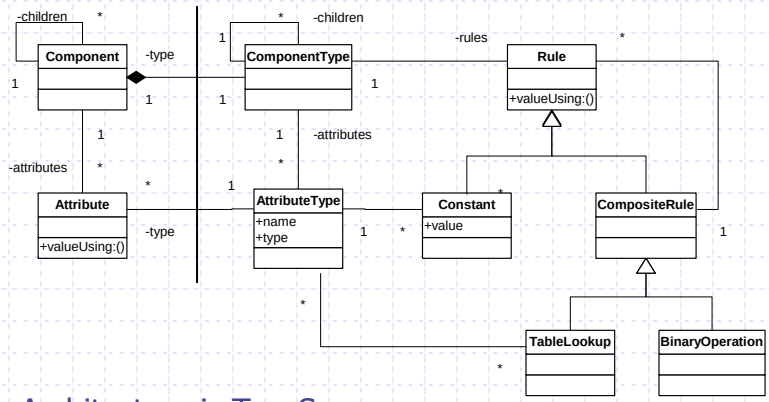
Wirfs-Brock's Policy

Policy
Auto
Collision
Auto
Home
Flood

New types of Policies require new classes and attributes…and changes to rules require updates to methods

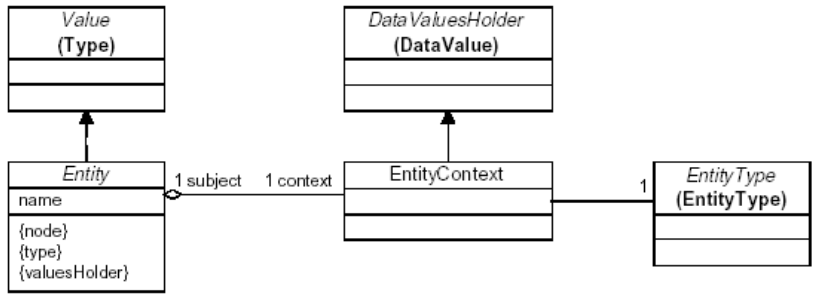
Slide - 91

User Defined Product
(insurance example)



Core Architecture is TypeSquare
Could quickly adapt to new rules within weeks or less
http://st-www.cs.illinois.edu/users/johnson/papers/udp/ Slide - 92

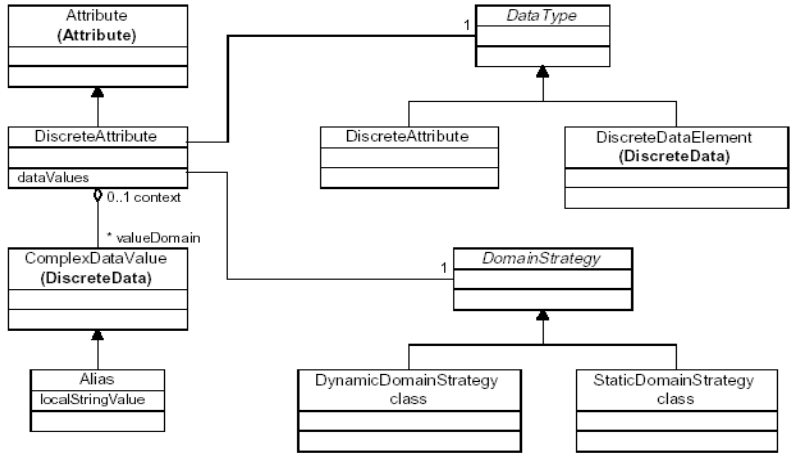
Objectiva Business Entities
(telephony billing system example)



Originally took a few person years to build.
AOM implementation reduced this to few person months

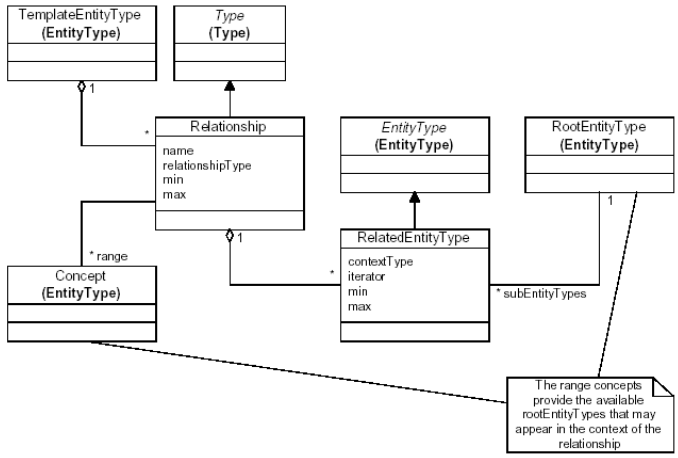
Slide - 93

Objectiva Attributes
(telephony example)



Slide - 94

Objectiva Entity Relationships
(telephony example)



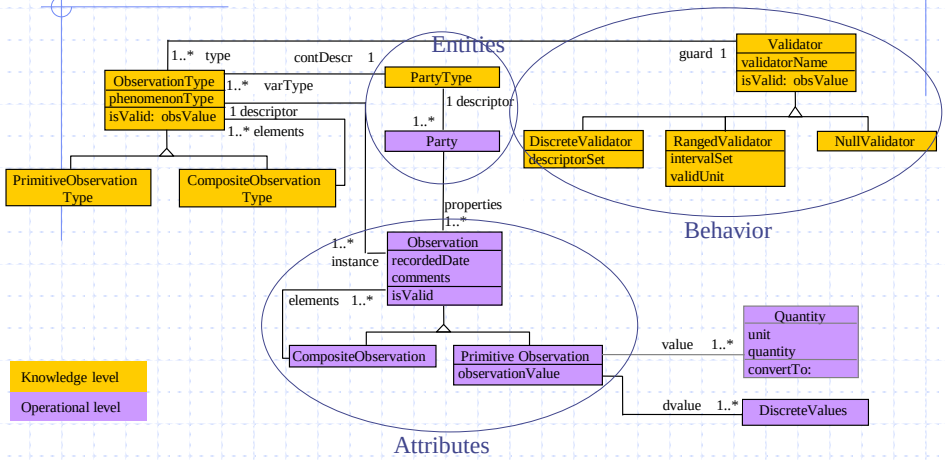
Slide - 95

How Objectiva Scaled

- Initial AOM implementation could handle ~1 million transactions per day
- Not enough for larger installations…
  - So, performance was tackled by adding caching and other point optimizations….increasing throughput to over 10 million transactions
  - Lesson: Meta Architectures can be tweaked for performance…nothing magic here

Slide - 96

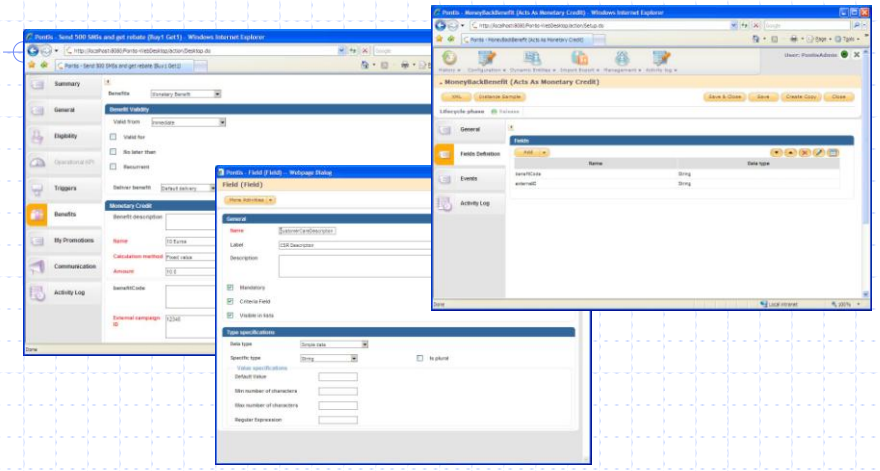
Medical Observations Design



Modelled100s of Entities with this core architecture.
Can add new entities or make changes to entities very quickly.

Slide - 97

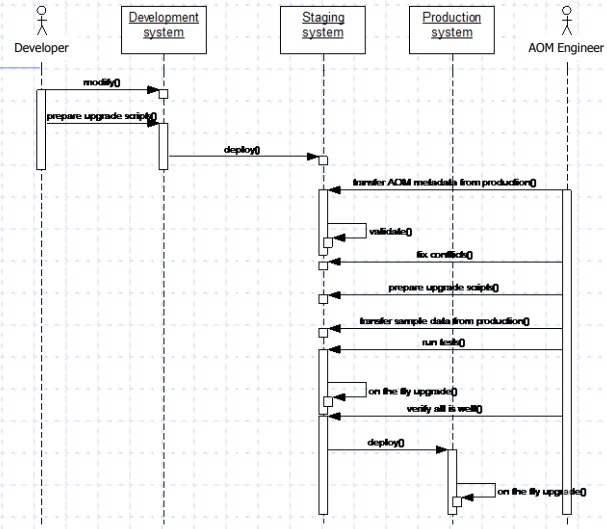
PONTIS AOM ENGINEERS



Telephony Marketing System
Highly Flexible

Slide - 98

PONTIS WORKFLOW



"Adaptive Object-Model Metadata Evolver", PLoP 2010, Atzmon Hen-Tov, David H. Lorenz, Lena Nikolaev, Lior Schachter, Rebecca Wirfs-Brock, Joseph Yoder Slide - 99

How the Pontis System Scales

- Reduced deployment from 1 year to 1 month
- Updates every couple of weeks
- Deployed to over 30 clients
- Code base much smaller 500,000 LOC vs 3,000,000 (previous implementation)
- Fastest growing business award
  - 5,500 percent within five years

Slide - 100

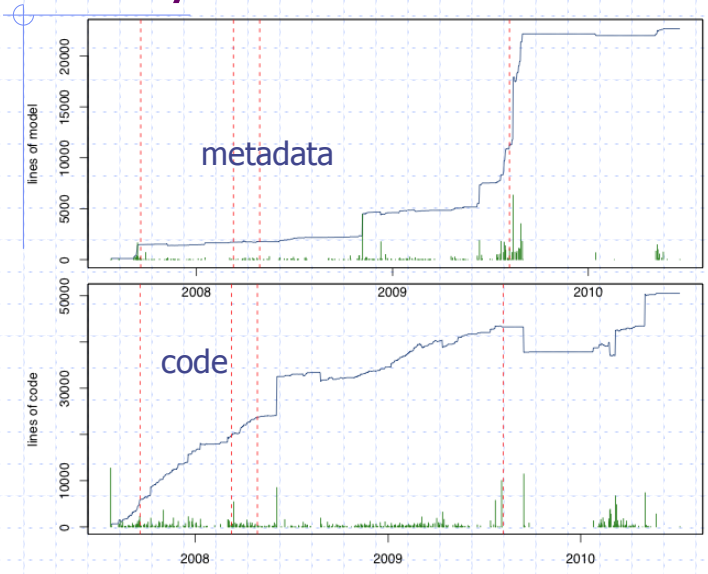
Portuguese Historical Artifacts Catalogue

- Catalogue of historical artifacts
- Mapping of site artifacts, relationships over time, historical site evolution
- 2 years churning on requirements
- Technical team finally made the decision to support easy ways to change the domain…metadata-driven software

1. Hugo Sereno Ferreira, Filipe Figueiredo Correia, Ademar Aguiar, and João Pascoal Faria. "Adaptive Object-Models: a Research Roadmap". International Journal On Advances in Software, volume 3, numbers 1 and 2, 2010. ISSN: 1942-2628.
2. Hugo Sereno Ferreira, Filipe Figueiredo Correia, Ademar Aguiar. "Design for an Adaptive Object-Model Framework: An Overview". Proceedings of the 4th International Workshop on Models@run.time. Co-located with the 12th International Conference on Model Driven Engineering Languages and Systems. Denver, Colorado. USA.
3. Hugo Sereno Ferreira, Ademar Aguiar, João Pascoal Faria. "Adaptive Object Modelling: Patterns, Tools and Applications". 3rd Symposium on Doctoral Students of Software Engineering. Proceedings of the 4th International Conference on Software Engineering Advances. Porto. Portugal.

Slide - 101

Project Timeline: Growth of the System

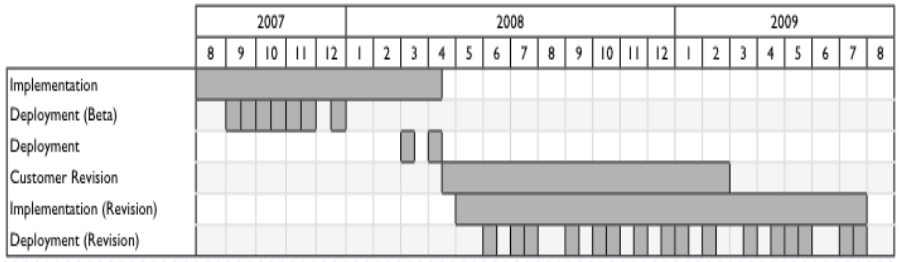


4 months to first deployment

features added rapidly at project end

Slide - 102

Project Timeline: Release Frequency



Once customers realized they could make model changes, they made lots of changes.

The rate of deployment of new releases increased, too.

Slide - 103

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

Slide - 104

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)

Slide - 105

When is an AOM 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

Slide - 106

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

Slide - 107

Some Disadvantages of Adaptive Object-Models

- Requires infrastructure for storing, building, and interpreting metadata.
- Requires strong design skills to create.
- Interpreting metadata can result in lower performance.

Slide - 108

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.

Slide - 109

Agile Best Practices for Developing Adaptive Systems

- Develop iteratively and incrementally.
- Get feedback early and often.
- Create flexibility only when and where needed.
- Develop tests for both the Object-Model and the Meta-Model.
- Support those who make changes and learn what they need (this may change over time).

Slide - 110

Summary

- Adaptive Object-Models are built upon domain expert knowledge and expose the elements of the domain and business rules.
- Best suited for systems that dynamically adapt to a changing (business) environment.
- Apply well-known design principles (e.g. TypeObject, Properties, Entity Relationship, and Strategies/RuleObjects).
- Take time to develop but can have **enormous** payoffs!

Slide - 111

Summary

- Separate what changes quickly from what changes slowly (hot-spots).
- AOM objects constitute a domain specific language.
- Building languages out of objects can be good…reflection guys say this!

Slide - 112

Summary

- AOMs support agile development:
  - Takes into account who changes what, when, and where.
  - Can be incrementally developed.
    - Start with "core" AOM, grow more as needed.
  - Empowers domain experts and gives them control over the system's domain evolution.
- Agile supports AOM development.
  - User scenarios to drive design.
  - Don't overdesign!

Slide - 113

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
- Atzmon Hen-Tov
- Ralph Johnson
- David H. Lorenz
- Lena Nikolaev
- Jeff Oaks
- Reza Razavi
- Nicolas Revault
- Dirk Riehle
- Lior Schachter
- Dave Thomas
- Michel Tilman
- Leon Welicki
- ...

Slide - 114

Resources

- Adaptive Object Models
  - www.adaptiveobjectmodel.com
- Agile Software
  - Agile Alliance: www.agilealliance.org
  - The Agile Manifesto
  - 12 Principles of Agile Development
  - Scrum Alliance: www.scrumalliance.org
  - Refactoring www.refactory.com

Slide - 115

That's All

Slide - 116