

A Framework for Financial Modeling

“a model to build a model”

Joseph W. Yoder
University of Illinois at Urbana-Champaign
The Refactory, Inc.
yoder@refactory.com
<http://www.refactory.com>

Sponsored by Caterpillar Inc. through the National Center for Supercomputer Applications

CATERPILLAR **NCSA™**

Copyright 2004 by Joseph W. Yoder 1



Goals

- Case study of developing a framework
- Case study of using design patterns
- Learn a framework for financial modeling

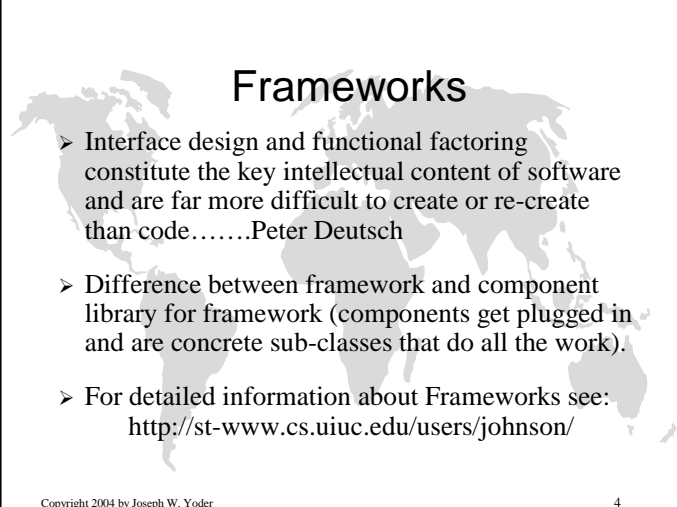
Copyright 2004 by Joseph W. Yoder 2



Overview

- Overview of Frameworks.
- What is a Financial Model?
- How we developed our framework.
- The Design of our framework.
- Patterns in our framework.

Copyright 2004 by Joseph W. Yoder 3



Frameworks

- Interface design and functional factoring constitute the key intellectual content of software and are far more difficult to create or re-create than code.....Peter Deutsch
- Difference between framework and component library for framework (components get plugged in and are concrete sub-classes that do all the work).
- For detailed information about Frameworks see:
<http://st-www.cs.uiuc.edu/users/johnson/>

Copyright 2004 by Joseph W. Yoder 4

Frameworks

- Frameworks solve a particular set of problems.
 - get different points of view
 - explain/defend current design
- Some frameworks are more technology (horizontal) frameworks verses application domain (vertical) frameworks.
 - Are we solving GUI's, object persistence verses more application domain related such as insurance or manufacturing.

Copyright 2004 by Joseph W. Yoder

5

Frameworks

- Framework is:
 - reusable design of an application or subsystem
 - represented by a set of abstract classes and the way objects in those classes collaborate.
- Use framework to build application by:
 - Creating new subclasses
 - Configuring objects together
 - Modifying working examples

Copyright 2004 by Joseph W. Yoder

6

Frameworks

- Framework prescribes how to decompose a problem.
- Not just the classes, but the way instances of the classes collaborate.
 - shared invariants that objects must maintain, and how they maintain them
 - framework imposes a collaborative model that you must adapt to.

Copyright 2004 by Joseph W. Yoder

7

Relevant Principles

- Frameworks are abstractions: people generalize from concrete examples
- Designing reusable code requires iteration
- Frameworks encode domain knowledge
- Customer of framework is application programmer

Copyright 2004 by Joseph W. Yoder

8

Generalize from Concrete Classes

- People think concretely, not abstractly.
- Abstractions are found bottom-up, by examining concrete examples.
- Generalization proceeds by:
 - finding things that are given different names but are really the same,
 - parameterizing to eliminate differences,
 - breaking large things into small things so that similar components can be found, and
 - categorizing things that are similar.

Copyright 2004 by Joseph W. Yoder

9

Finding Abstract Classes

- Abstract classes are discovered by generalizing from concrete classes.
- To give two classes a common superclass:
 - give them common interface
 - rename operations so classes use same names
 - reorder arguments, change types of arguments, etc.
 - refactor (split or combine) operations
 - if operations have same interface but different implementation, make them abstract
 - if operations have same implementation, move to superclass

Copyright 2004 by Joseph W. Yoder

10

Frameworks Require Iteration

Reusable code requires many iterations.

Basic law of software engineering:

“Johnson’s law”

If it hasn’t been tested, it doesn’t work.

Corollary: software that hasn’t been reused is not reusable.

Copyright 2004 by Joseph W. Yoder

11

White-box vs. Black-box

White-box ←————→ **Black-box**

Customize by subclassing

Emphasize inheritance

Must know internals

Simpler, easier to design

Harder to learn, requires more programming

Easier to customize because you can overwrite the code

Customize by configuring

Emphasize polymorphism

Must know interfaces

Complex, harder to design

Easier to learn, requires less programming

Harder to customize because you need to learn how objects collaborate

Copyright 2004 by Joseph W. Yoder

12

What is a Financial Model?

- Reports
- Answer “why”
- Correct errors, enter budget
- Depends on database
- Ensure security

Copyright 2004 by Joseph W. Yoder

13

Answering Why

- answers questions about finances
 - profit, return on assets
 - detailed costs
 - compare actual, budget, predicted
- high-level and detailed
- fixed reports and ad-hoc queries

Copyright 2004 by Joseph W. Yoder

14

What is a Financial Model?

- Business logic is equations
 - variable margin = net sales - variable cost
 - net sales = gross sales - warrantee
 - gross sales = sum *sales* column from *sales_and_transfer* table
- User interface just as important

Copyright 2004 by Joseph W. Yoder

15

Warning!
All numbers
are fake.

Top Level Dupont Model

Net Sales & Trans		Variable Margin		Operating Profit	
\$7,570	100.0%	(\$1,305)	(17.2%)	(\$1,871)	(25.3%)
\$17,100	100.0%	\$1,736	10.1%	(\$1,372)	(19.2%)
Variable Costs		Period Costs		Parts Profit	
\$8,870	117.2%	\$15,770	261.2%	\$7,650	101.2%
\$15,452	89.9%	\$35,109	192.6%	\$25,558	137.0%
Inventories		Current Assets		Other (Inc.) Exp.	
\$29,272	366.7%	\$29,272	366.7%	\$1,355	(18.4%)
\$107,362	824.6%	\$107,362	824.6%	(\$5,255)	(69.5%)
Receivables		Fixed Assets		Net Sales & Trans	
\$0	0.0%	\$54,000	721.2%	\$7,570	100.0%
\$0	0.0%	\$55,355	321.9%	\$17,190	100.0%
Other Assets		Total Assets		Asset Turnover	
\$0	0.0%	\$55,272	1107.9%		0.1
\$0	0.0%	\$102,637	946.2%		0.1

Copyright 2004 by Joseph W. Yoder

16

Inventories Drilldown

“Show calculation for Value”

	Budget	Actual	Profit +/-	% Change
Prime Products	\$3,273	\$80,857	\$77,585	2370.80%
Production Stores	\$26,000	\$26,525	\$525	2.02%
INVENTORIES	\$29,273	\$107,382	\$78,110	266.84%

From: January 1996 To: May 1996

Copyright 2004 by Joseph W. Yoder

17

Summary Report

Vehicles by marketing company.

Family Type	Model Number	Total	NACD	GACO	CAPLH	CBSA	CCL
HEX	245	(\$9,616)	(\$9,616)				
HEX	320	\$17,876,893	\$10,011,413	\$338,112	\$5,259,544	\$763,488	\$144,520
HEX	322	\$3,994,937	\$3,553,073				

From: January, 1995 To: December, 1995

Copyright 2004 by Joseph W. Yoder

18

Detailed Transactions

Inspect and edit the individual transactions.

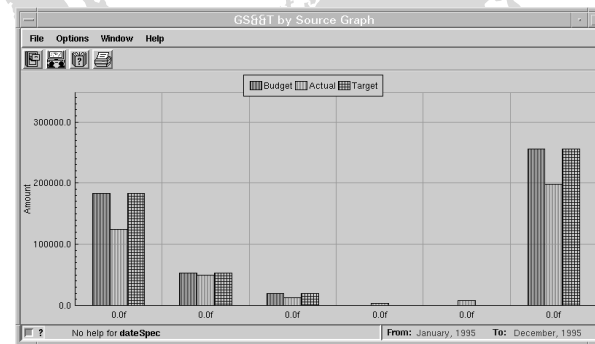
Date	Family	Section	Account	Seq Key	PCOS Actual	RD COST
Jan-96	DTH	05050	605	05	0	0
Jan-96	HEX	05050	606	05	0	0
Jan-96	LWL	05050	608	05	0	0
Jan-96	MWL	05050	608	05	0	0
Jan-96	SKD	05050	608	05	0	0
Jan-96	DTH	05050	607	04	0	0

From: January 1996 To: May 1996

Copyright 2004 by Joseph W. Yoder

19

Graphs



Copyright 2004 by Joseph W. Yoder

20

Summary of Reports

- Top level (Dupont or P&L)
- Drill down (ReportModel)
- Summary report
- Detailed transactions
- Graphs

Copyright 2004 by Joseph W. Yoder 21

Patterns for Developing Frameworks

- 1) Three Examples
- 2) White-box Framework
- 3) Component Library
 - Build applications and add components to library
- 4) Hot Spots
 - Separate Changeable from Stable Code
 - Design Patterns

Copyright 2004 by Joseph W. Yoder 22

Patterns for Developing Frameworks

- 5) Pluggable Objects
- 6) Fine-grained Objects
- 7) Black-box Framework
- 8) Visual Builder
- 9) Language Tools

<http://st-www.cs.uiuc.edu/users/droberts/evolve.html>

Copyright 2004 by Joseph W. Yoder 23

Three Examples

- Models for three business units
- Seemed completely different at first.
- Only one was fully implemented

Copyright 2004 by Joseph W. Yoder 24

White-box Framework

Five kinds of ApplicationModels,
with lots of subclasses

- DupontModel
- ReportModel
- DetailedModel
- SummaryModel
- GraphModel

Copyright 2004 by Joseph W. Yoder

25

User Interface Frameworks

- *DuPontModel* -
- *ReportModel* - Builds a spreadsheet interface using values and GUI descriptions from *ReportValues*.
- *SummaryReports*
- *DetailedWindows* - Edit and view individual transactions
- *GraphReports*

Copyright 2004 by Joseph W. Yoder

26

White-box Framework

- New window = new subclass
- Subclass has methods for
 - reading database
 - computing values
 - stuffing them in GUI
- Initialization registers with dependents

Copyright 2004 by Joseph W. Yoder

27

Component Library

- First, just abstract superclasses
- Second, query objects
- Third, GUI objects

Copyright 2004 by Joseph W. Yoder

28

Hot Spots

- Find aspects that change, and make them objects
- Often are patterns from *Design Patterns: Elements of Reusable Object-Oriented Software (GOF)*
- QueryObjects: Interpreter pattern

Copyright 2004 by Joseph W. Yoder 29

Interpreter Pattern

- Need to represent SQL to manipulate query:


```
SELECT SUM(sales) FROM sales_and_transfer
WHERE family='MWL' AND date >= '1/1/04'
AND date < '1/1/05'
```
- Problem: how do you represent a simple language?

Copyright 2004 by Joseph W. Yoder 30

Interpreter Pattern

- 1) make a class hierarchy that represents nodes in abstract syntax tree (SELECT, AND, <, tables, field names)
- 2) define methods to construct and manipulate tree
- 3) define method to compute value of query (this is the “interpreter”)

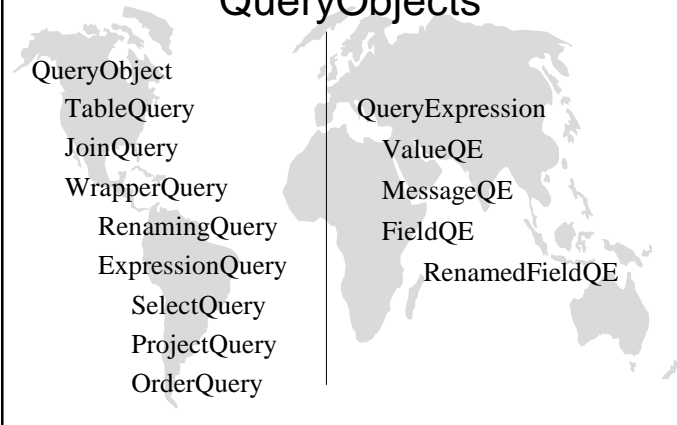
Copyright 2004 by Joseph W. Yoder 31

Instance Hierarchy

```
SELECT SUM(sales) FROM sales_and_transfer
WHERE family='MWL' AND date >= '1/1/04' AND date < '1/1/05'
```

Copyright 2004 by Joseph W. Yoder 32

QueryObjects



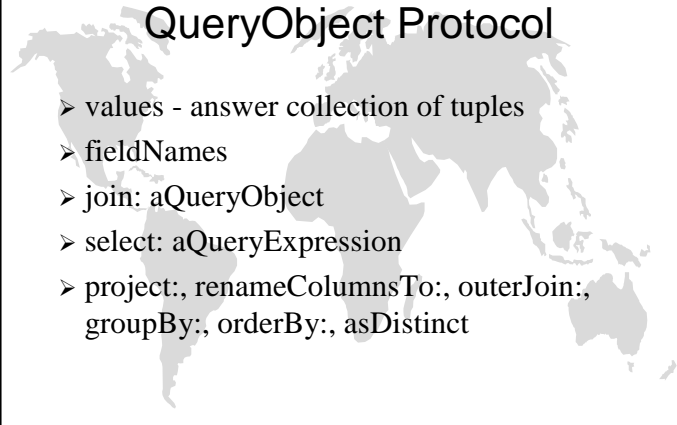
```
QueryObject
  TableQuery
  JoinQuery
  WrapperQuery
  RenamingQuery
  ExpressionQuery
  SelectQuery
  ProjectQuery
  OrderQuery

QueryExpression
  ValueQE
  MessageQE
  FieldQE
  RenamedFieldQE
```

Copyright 2004 by Joseph W. Yoder

33

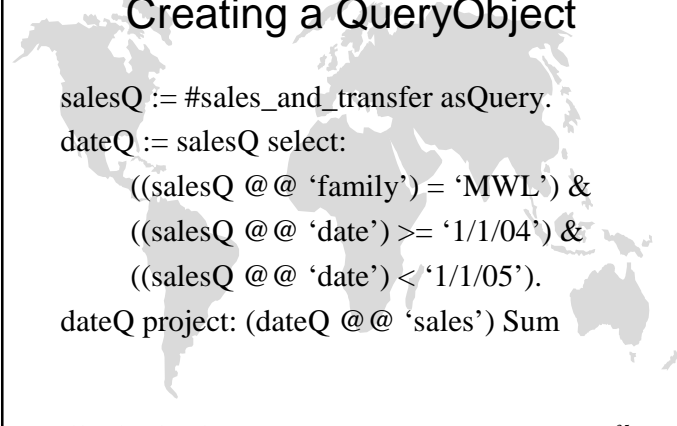
QueryObject Protocol

- 
- values - answer collection of tuples
 - fieldNames
 - join: aQueryObject
 - select: aQueryExpression
 - project:, renameColumnsTo:, outerJoin:, groupBy:, orderBy:, asDistinct

Copyright 2004 by Joseph W. Yoder

34

Creating a QueryObject



```
salesQ := #sales_and_transfer asQuery.
dateQ := salesQ select:
  ((salesQ @@ 'family') = 'MWL') &
  ((salesQ @@ 'date') >= '1/1/04') &
  ((salesQ @@ 'date') < '1/1/05').
dateQ project: (dateQ @@ 'sales') Sum
```

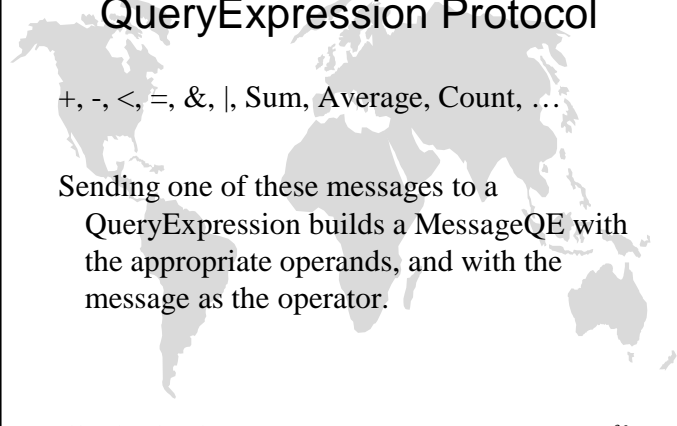
Copyright 2004 by Joseph W. Yoder

35

QueryExpression Protocol

+, -, <, =, &, |, Sum, Average, Count, ...

Sending one of these messages to a QueryExpression builds a MessageQE with the appropriate operands, and with the message as the operator.



Copyright 2004 by Joseph W. Yoder

36

Leading to Black-box

- Component Library
- Hot Spots
- Pluggable Objects
- Fine-grained Objects
- Black-box Frameworks

Copyright 2004 by Joseph W. Yoder

37

First Design

- Class hierarchy of ReportModels, ReportModel creates QueryObjects.
- Improvement: separate logic and GUI
- Two hierarchies: ReportModel and ReportValues.
 - Result: twice the classes, some reuse

Copyright 2004 by Joseph W. Yoder

38

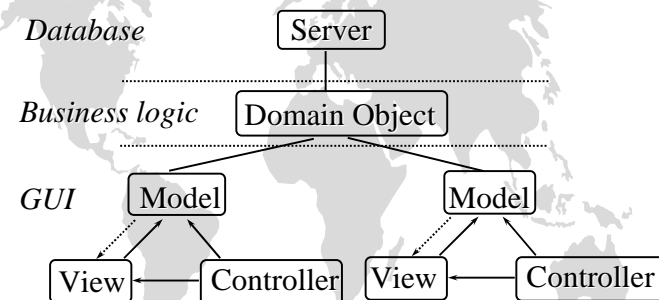
First Separation

- | | |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <ul style="list-style-type: none"> ➤ ReportModel <ul style="list-style-type: none"> ▪ SalesModel ▪ InventoryModel ▪ ... <p>Uses QueryObjects</p> | <ul style="list-style-type: none"> ➤ ReportValues <ul style="list-style-type: none"> ▪ SalesValues ▪ InventoryValues ▪ ... <p>Makes QueryObjects</p> |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

Copyright 2004 by Joseph W. Yoder

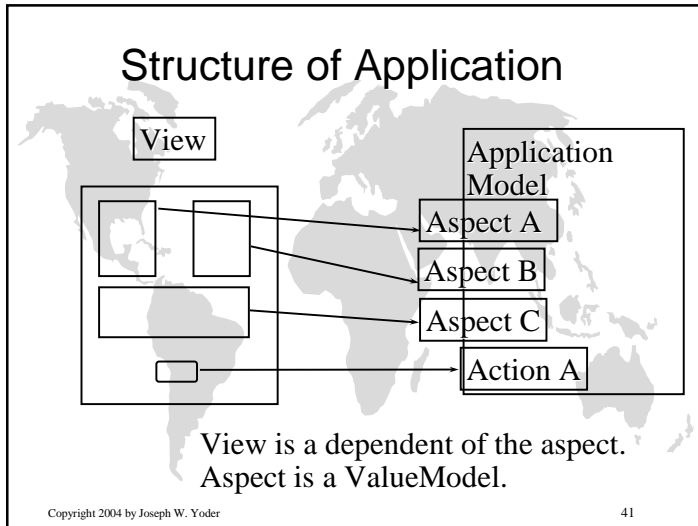
39

Three-tiered Client-Server

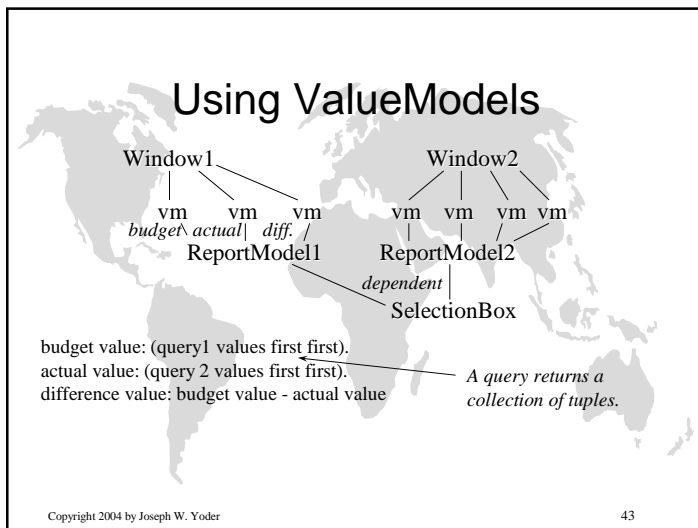


Copyright 2004 by Joseph W. Yoder

40



- ### ValueModel
- Example of Observer pattern
 - View (observer) registers with ValueModel (subject) and is notified when it changes.
 - ValueModel protocol
 - value, value:
 - addDependent:, removeDependent
 - Observer protocol
 - update:
- Copyright 2004 by Joseph W. Yoder 42

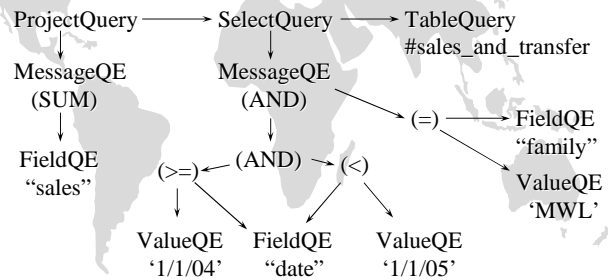


- ### Alternative Solutions
- ReportModel depends on SelectionBox.
 - updates for too many changes
 - ReportModel depends on ValueModels from SelectionBox used in QueryObject
 - hard to manage dependencies
 - ReportModel depends on QueryObject, QueryObject depends on ValueModels from SelectionBox
- Copyright 2004 by Joseph W. Yoder 44

Observer and QueryObjects

Let ValueQE refer to a ValueModel.

Let each QueryObject observe its components.



Copyright 2004 by Joseph W. Yoder

45

Old way - route change through report

budget value: (query 1 values first first).

actual value: (query 2 values first first).

difference value: budget value - actual value

Update

New way - route change directly to ValueModel

budget := QueryHolder on: query1

actual := QueryHolder on: query2

difference := budget - actual

Initialization

Requires:

QueryHolder - adapts QueryObject to ValueModel

ValueModel understands +, -, *, /, etc

Copyright 2004 by Joseph W. Yoder

46

QueryHolder

Adaptor pattern - subclass of ValueModel that lets QueryObject act like ValueModel.

instance variables: query, values

query: aQuery

query := aQuery.

aQuery addDependent: self

Copyright 2004 by Joseph W. Yoder

47

QueryHolder

update

values := aQuery values

self changed

value

^values first first

Copyright 2004 by Joseph W. Yoder

48

Arithmetic on ValueModels

ValueModel implements arithmetic by creating ValueModels that compute function.

+ anObject

^BlockValue

on: [:a :b | a value + b value]

with: (Array with: self with: anObject)

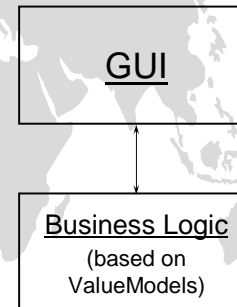
Copyright 2004 by Joseph W. Yoder

49

Result of Refactoring

Reuse GUIs, change ValueModels.

Hard part is creating ValueModels and connecting them to GUI.



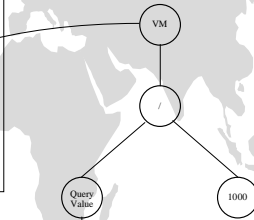
Copyright 2004 by Joseph W. Yoder

50

Typical Values in a Report

Sales Report		
budget	actual	variance
	\$ 5,000	

Thousands of Dollars



Select all sales from North America for the specified date and products which returns \$5,000,000

Copyright 2004 by Joseph W. Yoder

51

Business logic is equations expressed with ValueModel and QueryObjects

- ❖ Values = functions of other values
- ❖ Values = queries from the database

variable margin = net sales - variable cost

net sales = gross sales - warrantee

gross sales = sum *sales* column from *sales_and_transfer* table

Copyright 2004 by Joseph W. Yoder

52

Problems

- How do we go from one report to the next?
- How do we connect report to business model?
- Must define business model flexibly
- Must define GUI flexibly

Copyright 2004 by Joseph W. Yoder

53

Specifications

- A ReportSpec
 - has name
 - has parameters
 - has menus, which name other reports
- DetailedReportSpec and SummaryReportSpec are parameterized with QueryObjects.
- GraphReportSpec is parameterized with ValueModels

Copyright 2004 by Joseph W. Yoder

54

ReportValues

Many tables.
Many columns
Each table has a sequence of valueModels
Total at end.

Period Costs				
File Graph Formula Detailed Summary Variance Window Help				
R & D				
	Budget	Actual	Profit +/-	% Change
Internal R&D	\$1,796	\$5,342	(\$3,546)	(197.40%)
Inbound R&D	\$1,235	\$5,263	(\$4,028)	(326.01%)
Outbound R&D	(\$51)	(\$1,251)	\$1,200	(232.10%)
Total R&D	\$2,980	\$9,344	(\$6,364)	(213.55%)
Office Costs				
Commercial & General	\$916	\$815	\$103	11.27%
Planning	\$2,560	\$2,755	(\$195)	(7.61%)
Internal Services	\$3,197	\$2,688	\$509	16.55%
Inbound Services	\$757	\$3,528	(\$2,771)	(369.11%)
Outbound Services	\$82	\$452	(\$370)	(450.55%)
Total Office Costs	\$7,515	\$10,219	(\$2,704)	(35.98%)
Factory Costs				
Depreciation	\$411	\$5,073	(\$4,662)	(1134.36%)
Occupancy Costs	\$1,489	\$1,660	(\$170)	(11.45%)
Mach/Equip. Repair	\$1,601	\$1,242	\$359	22.40%
Other Prod. Costs	\$744	\$5,571	(\$4,827)	(648.77%)
Total Factory Costs	\$4,245	\$13,546	(\$9,301)	(219.11%)
PERIOD COSTS	\$14,739	\$33,109	(\$18,369)	(124.63%)

Copyright 2004 by Joseph W. Yoder

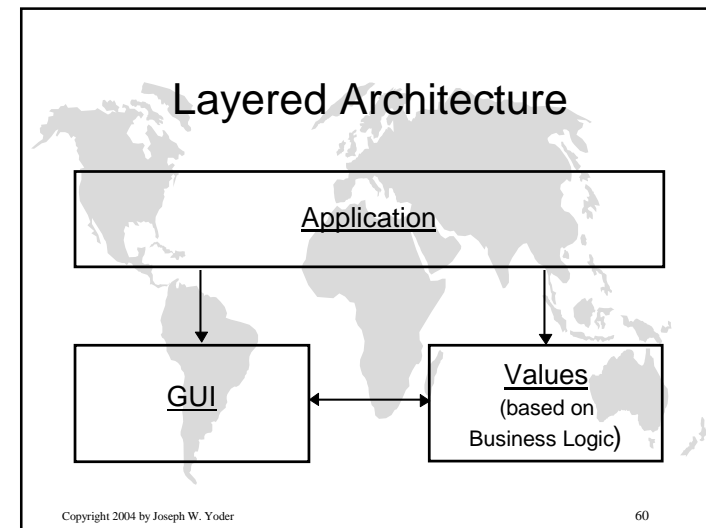
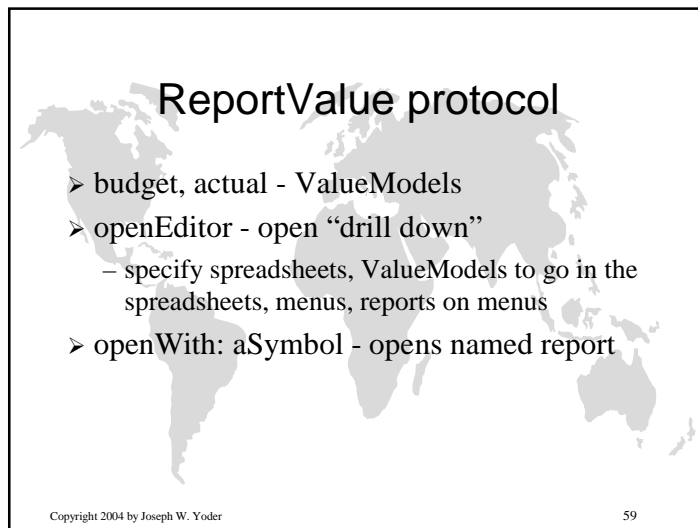
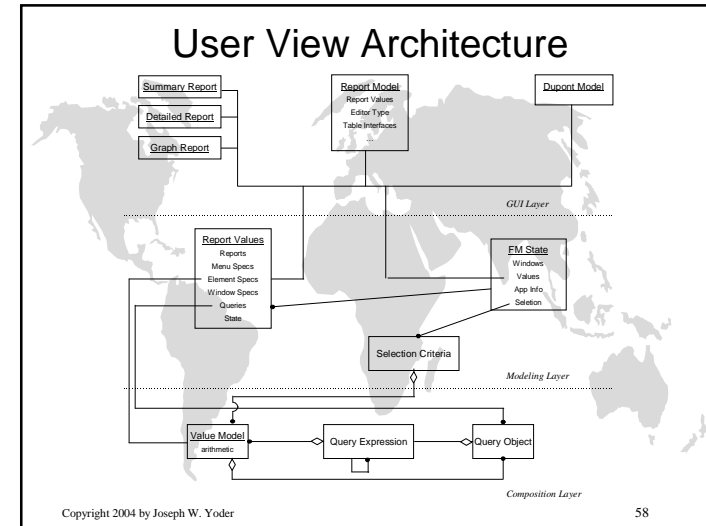
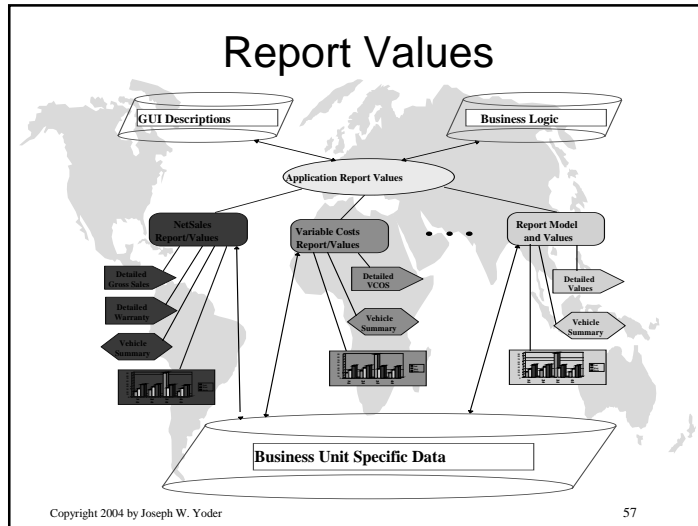
55

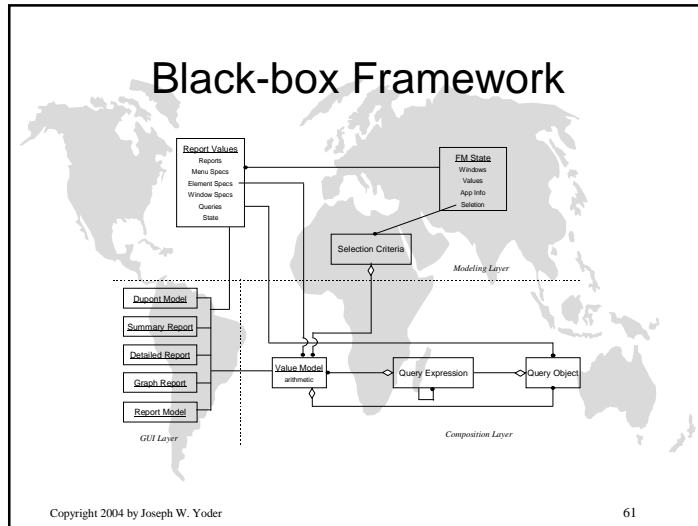
Solution

- ReportValues responsible for
 - knowing values
 - knowing how to compute values
 - knowing how to display more detail (drill-down) on values
- Top level starts up ReportValues which starts up next.

Copyright 2004 by Joseph W. Yoder

56





Visual Builder

- Make a GUI to define Specs.
- This GUI is a language for defining financial models.

Copyright 2004 by Joseph W. Yoder 62

Builders

- Equations in a ReportValue
 - expressions
 - queries
- GUIs
 - ReportValue (Drill down)
 - Graphs - specify business logic, labels
 - Detailed - specify query, labels, editing
 - Summaries - specify query, grouping, columns to sum and calculate
- Selection

Copyright 2004 by Joseph W. Yoder 63

Language Tools

- Languages need debuggers, profilers, version control, etc.
- Built some core GUI's for describing the business rules with some checkers for verifying the rules.

Copyright 2004 by Joseph W. Yoder 64

Summary of Architecture

- Builders
- ReportValues, Selection Criterion, FMState
- GUI frameworks
- ValueModel, QueryObject

Copyright 2004 by Joseph W. Yoder

65

Summary of Architecture

- business model is not object-oriented, just a bunch of equations
- object model is the language for specifying business model, not the business model

Copyright 2004 by Joseph W. Yoder

66

Data Model

- Application Specific
 - holds “real data”
 - changes with every business model
- Generic
 - specifies business logic and GUI
 - never changes

Copyright 2004 by Joseph W. Yoder

67

Other Features

- Any window can print itself
- Automated testing support
- Security for editing and/or viewing the data; configured by administrators.

Copyright 2004 by Joseph W. Yoder

68

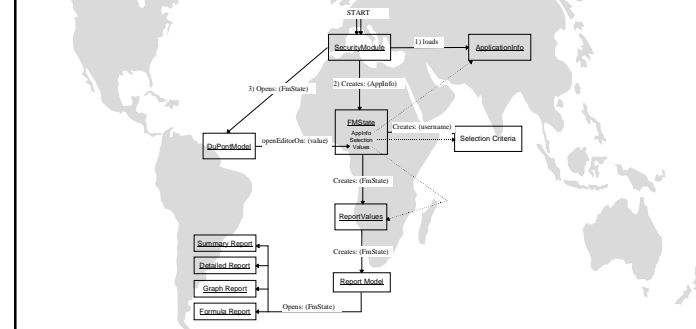
Security Requirements

- Control passwords
- Control login
- Users have roles
- Role can only view a specified list of products.
- Role can only edit a subset of the specified list of products.
- All security features can be controlled by administrators

Copyright 2004 by Joseph W. Yoder

69

FM Creational Diagram



Copyright 2004 by Joseph W. Yoder

70

How to Develop a New Financial Application

- Analyze business unit
- Build business-unit data model
- Specify GUI and business logic
- Install and Test

Copyright 2004 by Joseph W. Yoder

71

Analyze Business Unit

- Questions to ask a new business unit
 - Values to be calculated (netsales, vcos, pcos, ...)
 - User interface
 - top level
 - Drill Downs (summary and detailed)
 - Graphs of values
 - Error-Correction/Analysis modules

Copyright 2004 by Joseph W. Yoder

72

Some of the Patterns Used

- Builder
- Interpreter
- Model-View-Controller
- Reports
- Adapter
- Command
- Observer
- Decorator
- Visitor
- Singleton
- Factory Method
- Constraints
- Null Objects
- Composite

Copyright 2004 by Joseph W. Yoder 73

Summary

- We have developed a reusable design for financial applications
- Domain specific “Visual-Language”
- Framework emerges by repeatedly refactoring system to eliminate complexity and create flexibility

Copyright 2004 by Joseph W. Yoder 74

Related Links

- The following link discusses the details of the framework
http://www.joeyoder.com/financial_framework
- Good Object-Oriented page with framework references
<http://st-www.cs.uiuc.edu/users/johnson/>
- The Evolutionary Patterns Paper - PLoP '96
<http://st-www.cs.uiuc.edu/users/droberts/evolve.html>
- The Reporting Patterns describing Query-Objects - PLoP '96
<http://www.joeyoder.com/papers/patterns/Reports/>

Copyright 2004 by Joseph W. Yoder 75

Related Links

- The security patterns used in this framework - PLoP '97
<http://www.joeyoder.com/papers/patterns/Security/>
- Dmitry Zelenko's Masters Thesis describing Query Models
<http://www.joeyoder.com/papers/thesis/zelenko.ps/>
- Reflective Facilities and Evolutionary approaches - PLoP '95
<http://www.joeyoder.com/papers/patterns/Evolution/>
- Jeff Barcalow's Masters Thesis describing Scenario Planning
<http://www.joeyoder.com/papers/thesis/barcalow.html>
- Adaptive Object-Model Architecture
<http://www.adaptiveobjectmodel.com>

Copyright 2004 by Joseph W. Yoder 76