Data Upgrade as a First-Class Citizen

Atzmon Hen-Tov
VP R&D - Pontis

Saturn 2015
Baltimore, MD - April 27-30, 2015
Pontis is a developer of iCLM™
An Online Marketing Automation product for communication service providers

Our customers are Tier I operators in Europe, CIS, South-America, Africa and the Far east

The product engages the customers proposing products and services that best suit the individual customer based on analysis of the customer behavior

These products conform to strict non functional requirements (Telco grade) supporting billions of transaction per day

The dynamics of the marketing domain requires high agility from the product
Problem Statement

Many production systems

Different development teams

Multiple technologies

We need to upgrade frequently, we cannot risk data loss!
Limitation of Existing Solution

We followed Continuous Integration best practices and recipes

Develop Source Control Build Test

BUT... we found the following deficiencies:

- You might miss changes not covered by tests
- You are counting on developers awareness
- DBA involvement hampers agility
- Developers cannot master all technologies
- Developers tend to avoid refactoring after a feature is shipped
- In NoSQL, lack of schema further intensifies the risk (Polyglot persistence)
Outline of Our Solution

1. **Validation** - The IDE alerts the developers on changes that they didn’t handle

2. **Data Upgrade DSL** - Upgraders are written in a DSL abstracting the underlying technologies and supporting multiple technologies simultaneously
Background: High Level Architecture

Integration Layer

- RT Complex Event Processing
- Subscriber Data Store

User Interface

- Setup
- CSR
- Monitoring
- Marketing

Operator Systems

- OCS
- IN
- CDR
- PCC
- CRM

Channels

- Business Discovery, Monitoring & Reporting

Decisioning Engine

Subscriber Profile

Analytics

DWH
Background: Variability between customers

Multiple, highly customized instances exist for each customer and need to be supported.
Over the last 2 years we evolved our architecture to **Big-Data**
Background: Supporting Multiple Technologies

Now we support both legacy and Big-Data

...and permutations of thereof
Background: Supporting Multiple Technologies

Now we support both legacy and Big-Data

...and permutations of thereof

Multiple storage technologies and data formats

Entities are mapped to several store technologies
Problem Statement

Many production systems

Different development teams

Multiple technologies

We need to **upgrade frequently**, we cannot risk data loss!
Our Solution: IDE support

Developer **changes structure** of persistent classes

IDE **updates** the **model snapshot**

IDE **compares** with model snapshot of previous version

Uncovered upgraders are reported to the Developer

Developer writes upgrade-DSL declarations and possibly **custom upgraders**

IDE searches for upgraders declared for each diff

IDE validation is satisfied

IDE compares with model snapshot of previous version

Uncovered upgraders are reported to the Developer
Our Solution: IDE support

Developer changes structure of persistent classes

IDE updates the model snapshot

IDE compares with model snapshot of previous version

The DSL execution generates the concrete upgrade scripts for all relevant store technologies

IDE searches for upgraders declared for each diff

The IDE validation is satisfied

Uncovered upgraders are reported to the Developer

At runtime:
- Run the relevant scripts during installation
- Activate online upgraders during object deserialization

Developer writes upgrade-DSL declarations and possibly custom upgraders

UPGRADE!

The DSL execution generates the concrete upgrade scripts for all relevant store technologies

PONTIS
Our Solution: Upgrade DSL

Offer
- allowance: double
- message: string
- price: double
- validity: date

Service
- listPrice: double

message = 6 hours of long distance calls for just $2
Our Solution: Upgrade DSL

Model evolution – Adding numeric discount property

Offer
- allowance: double
- message: string
- price: double
- validity: date
- discount: double

Service
- listPrice: double

Upgrade

UpgradeSpec {
  ChangeSpec className="ClassChange" {
    className="Offer"
    Changes {
      Change class="PropertyAdded" {
        propertyName="discount"
        defaultValue=0
      }
    }
    Upgraders {
      Upgrader class="AutoUpgrader" {}
    }
  }
}
Our Solution: Upgrade DSL

Upgrade Language Meta Model

- **UpgradeSpec**
  - changeSpec

- **ClassChange**
  - className: string
  - superclass: Class

- **PropertyChange**
  - propertyName: string

- **ClassRenamed**
- **ClassMetadataChanged**
- **PropertyAdded**
  - propertyType: Type
  - defaultValue: string
- **PropertyTypeChanged**
  - propertyType: Type
- **PropertyMetadataChanged**
- **Class Super Change**
- **Class Added**
- **Class Removed**
- **PropertyRemoved**
- **Property Renamed**
Our Solution: Upgrade DSL

Upgrade Language Meta Model

- **Upgradespec**
  - Provides default behavior for the upgrade (such as adding a column and populating it with the default value)

- **Upgrader**
  - Allows for custom SQL/Hbase/etc., script when the auto-upgrader behavior is not sufficient.
  - Translates an old instance to valid instance(s) in the new model. This is done online during deserialization

- **StoreType**

- **AutoUpgrader**
  - Script: string

- **OnlineUpgrader**
  - upgradeInstance(Map< String, Object >): Object
Our Solution: Upgrade DSL

Model evolution – Custom upgrader for the discount property

**Offer**
- allowance: double
- message: string
- price: double
- validity: date
- discount: double

**Service**
- listPrice: double

```java
public class MyOfferUpgrader extends OnlineUpgrader {
    Object upgradeInstance(Map<String, Object> oldInstance) {
        Offer result = new Offer();

        // Copy all unchanged Offer fields from the map to the new instance.
        super.copyAllFields(oldInstance, result);

        // Calculate the discount.
        result.setDiscount(result.getService().getListPrice() - result.getPrice());

        return result;
    }
}
```
Our Solution: Upgrade DSL

Model Evolution – Discounters added to the model. “discount” is changed to be of type Discount

<table>
<thead>
<tr>
<th>Offer</th>
<th>Service</th>
</tr>
</thead>
<tbody>
<tr>
<td>allowance: double</td>
<td>listPrice: double</td>
</tr>
<tr>
<td>message: string</td>
<td></td>
</tr>
<tr>
<td>price: double</td>
<td></td>
</tr>
<tr>
<td>validity: date</td>
<td></td>
</tr>
<tr>
<td>discount: double</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Discount</th>
<th>Service</th>
</tr>
</thead>
<tbody>
<tr>
<td>FixedAmountDiscount</td>
<td>listPrice: double</td>
</tr>
<tr>
<td>PercentageDiscount</td>
<td></td>
</tr>
<tr>
<td>ConditionedDiscount</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Condition</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ConditionedDiscount</td>
<td></td>
</tr>
</tbody>
</table>

Upgrade

```
UpgradeSpec {
    ChangeSpec class="ClassAdded" {
        className="Discount"
    }
    ChangeSpec class="ClassAdded" {
        className="FixedAmountDiscount"
    }
    ChangeSpec class="ClassChanged" {
        className="Offer"
        Changes {
            Change class="PropertyTypeChanged" {
                propertyName="discount"
                newType="Discount"
            }
        }
    }
    Upgraders {
        Upgrader class="AutoUpgrader" {
            // transform the numeric discount into
            // an instance of FixedAmountDiscount
            Upgrader class="MyOfferDiscounterUpgrader" {
                java_package = "com.pontis.upgraders"
            }
        }
    }
}
```
Related work

Database Continuous Integration
- [Sadalage 2007] Recipes for Continuous Database Integration
- [Fowler & Sadalage 2012] NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence
- [Humble, Farley] Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation

Evolution patterns
- Fowler, Martin; Sadalage, Pramod J. Evolutionary Database Design http://martinfowler.com/articles/evodb.html

Domain Specific Language Workbench
- The Ink language workbench: https://code.google.com/a/eclipselabs.org/p/ink/
Conclusion

• **Problem**
  • No support for Data upgrade in development environment
  • Current practices rely on developer awareness

• **Solution – Technique for handling data upgrades using tool support and DSLs**
  • IDE alerts developers at design time on model changes
  • Simplified handling of data upgrade with DSLs

Pontis has implemented the solution and gathered experience from hundreds of SW upgrades in numerous Telco production sites

• **Our (subjective) assessment:**
  • Enhanced agility
  • Less data upgrade issues leak to productions
  • Developers are not avoiding refactor
Thank You!

Atzmon Hen-tov

Atzmon@ieee.org
How do we tackle both variability and agility

To address the variability challenges our development is based on:

**Layered pipeline**

**Executable Model**

**AOM**

**ModelTalk:** An interpretive domain specific language workbench

IDE: DSL and Java integrated development environment

Runtime platform

DSL: Domain-specific language
How do we tackle both variability and agility

To address the variability challenges our development is based on:

<table>
<thead>
<tr>
<th>Layered pipeline</th>
<th>Executable Model</th>
<th>AOM</th>
</tr>
</thead>
</table>

**Modeltalk**: An interpretive domain specific language workbench
How do we tackle both variability and agility

To address the variability challenges our development is based on:

- **Layered pipeline**
- **Executable Model**
- **AOM**

**R&D Core Layer**

**R&D Branching Layer**

**Project Specific IT Layer**

- **Operator 1**
  - Shared Assets
  - Particular Assets
  - Production Operator 1

- **Operator 2**
  - Shared Assets
  - Particular Assets
  - Production Operator 2

- **Domain A Branch**
  - Shared Assets

- **Domain B Branch**
  - Shared Assets

**Shared Assets**
How do we tackle both variability and agility

To address the variability challenges our development is based on:

Layered pipeline  Executable Model  AOM

A strong metadata layer empowering users to evolve the system (see more in http://adaptiveobjectmodel.com/)
Company

- Healthcare analytics company that provides insights into the value of medical technologies
- Small, cross-functional team
Platform

- Analytics platform that allows analysts to complete projects 90% faster than with traditional approaches

- Users can implement a broad spectrum of data analytics projects

- Currently supports 12 types of datasets
Users

- Research analysts and managers
- Life Sciences
  - Pharma
  - Biotech
  - Medical devices
- Health Plans
- Medical Groups
The Data

- Data classes are electronic health records, administrative claims, hospital billing/clinical data, survey data

- Within each class there are many different specific data sources provided by commercial companies or government agencies

- Relational schema or denormalized schema (sometimes mixtures of both with schema varying year-by-year)

- Datasets have lots of variation in terms of files (tables) provided/fields available
The Data, cont’d.

● “Large” datasets
  ○ 150 million patients
  ○ 100 billion rows
  ○ 6TB

● Statistically De-Identified
Data Model

- Demographics
- Enrollment
- Encounter
- Medication
- Diagnosis
- Procedure
- Lab
- Observation
- PhysicianNote
# Example Projects

<table>
<thead>
<tr>
<th>Therapeutic Area</th>
<th>Data Source</th>
<th>Analysis Topic</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atrial Fibrillation</td>
<td>EMR</td>
<td>Matched cohort analysis of resource use and outcomes</td>
<td>1 meeting abstract 1 manuscript in progress</td>
</tr>
<tr>
<td>Diabetes</td>
<td>EMR</td>
<td>Treatment patterns and outcomes</td>
<td>Study report</td>
</tr>
<tr>
<td>Epilepsy</td>
<td>Claims</td>
<td>Adverse events, adherence, and adjunctive therapy</td>
<td>4 meeting abstracts</td>
</tr>
<tr>
<td>Multiple Sclerosis</td>
<td>Claims</td>
<td>Prevalence and treatment rates/outcomes</td>
<td>3 meeting abstracts</td>
</tr>
<tr>
<td>Orthopedic Surgery</td>
<td>Hospital</td>
<td>Matched case control analysis of costs</td>
<td>3 meeting abstracts</td>
</tr>
<tr>
<td>Stroke</td>
<td>Claims</td>
<td>Risk factors and costs</td>
<td>2 meeting abstracts 1 manuscript 1 manuscript in progress</td>
</tr>
<tr>
<td>Venous Thromboembolism</td>
<td>EMR</td>
<td>Treatment patterns and Outcomes</td>
<td>1 meeting abstract 1 manuscript</td>
</tr>
</tbody>
</table>
Example Projects, cont’d.

Treatment Patterns and Outcomes Among Hospitalized Patients With Venous Thromboembolism in the United States: An Analysis of Electronic Health Records Data

Abstract

Background: With the advent of new treatment options for venous thromboembolism (VTE), it is valuable to gain insights into current clinical practice. Objective: Assess treatment patterns and outcomes among patients hospitalized for VTE. Methods: This retrospective study evaluated patients hospitalized with an incident VTE diagnosis (index) from 2008 to 2012 in a de-identified electronic health record database. Patients were further required to receive anticoagulant treatment and/or a VTE-related procedure for study inclusion. Patients were excluded if they: (1) did not have a medical encounter in the 6 months before index (baseline); (2) had a prior VTE diagnosis or used an anticoagulant during the baseline period; or (3) had a diagnosis of atrial fibrillation/thrombus, cardiomyopathy, or a coagulation disorder during baseline or the year after index (follow-up). Hospitalizations for recurrent VTE and bleeding were evaluated. Results: A total of 2,000 patients were identified (mean age, 65.9 years; 53.0% women), with a mean length of stay of 8.1 days. Of the VTE types, acute DVT was the most common (41.9%), followed by PE (31.3%), and DVT + PE (34.3%). Almost all patients (96.9%) received anticoagulants, of whom 94.6% received heparin and 91.5% received warfarin. Although 77.4% of warfarin users were prescribed at discharge, only 40.2% had a warfarin prescription within 30 days of discharge. Overall, 30-day, 90-day, and 1-year VTE recurrence rates were 1.8%, 4.2%, and 7.9%, respectively, and the major bleeding rate was 6.9%. Conclusions: In a real-world population of hospitalized VTE patients, hospital treatment in combination with warfarin was common. However, continuation of warfarin post-discharge was challenging, limiting the ability to improve continuation of therapy and to reduce VTE recurrence. Keywords: treatment; thromboembolism; anticoagulation; adverse outcomes

Introduction

Venous thromboembolism (VTE), including deep vein thrombosis (DVT) and pulmonary embolism (PE), has an estimated annual prevalence of >400 cases per 100,000 patients in the United States, a figure that is expected to increase because of the documented increased risk of VTE with increasing age as well as the aging of the US population.4 Venous thromboembolism poses a significant clinical and economic burden to those who are affected. The 1-year mortality rate after a VTE episode is estimated to be 17% to 23%, and, after a diagnosis of VTE, patients are prone to other complications such as recurrent VTE, major bleeding, heparin-induced thrombocytopenia.
# Workflows

## Old Architecture

1. **Written Specs**: 2 weeks
2. **Initial Custom Coding**: 2 weeks
3. **Revise Code**: 2 weeks
4. **Final Results**

## New Architecture

1. **Written Specs**: 1 Day
2. **Initial Project Creation**: 2 Days
3. **Revise Project**: 1 Day
4. **Final Results**
Demo

- Presented live at SATURN 2015
Data Model

● Adaptable Core

● Extensible

● Adapt or Extend?
Specification Language

● Users need a way to specify types of analyses they want to perform

● Can be defined using the web interface

● Can be submitted programmatically through a REST API

● “Internal” DSL
Code Generation

- Templates/Model Interpretation
- Complex, optimized code generated based on user specs
Web App

- No coding for end user
- Common UI for many data sources
- Concepts from domain model apply fairly directly
Key Flex Points

● Supporting Additional Data Classes

● Adding Patient Population Filters (Engineering)

● Adding Measures (Engineering)

● Inline Custom Measures (User/Engineering)

● Integration with R, SAS, other tools (User)
Lessons from Building Domain-Specific Analytics Platform

- Execution and programming time savings
- Higher quality
- Difficult to create appropriate abstractions to support variability
- Flexibility versus ease of use
Lessons, cont’d.

● 80/20 Rule

● Flex points

● Core Model
  ○ Adaptable
  ○ Extensible
Sustainable Architecture
“Supporting Data Variability”

Adaptive Object Models & Meta Modeling

SATURN – April 28th, 2015
Joseph W. Yoder -- www.refactory.com

Copyright 2014 Joseph W. Yoder & The Refactory, Inc.
Sustaining Architecture

When dealing with data variability...

“What can be done to help sustain our architecture in the long run?”
Motivation: Need to Quickly Adapt to Change

- Business Rules and 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.
Core Ideas

- Separate what changes from what doesn’t change
- Protect core components
- Find the variability points and provide ways to support this variability in data
  - Description of changes (metadata)
  - Hook Points for variability
  - Dynamic Process Rules with Data
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*
Sustaining Your Architecture

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.

- The platform
- Infrastructure
- Data schema
- Standard frameworks, components, services
- Abstract classes and interfaces
- Classes
- Code
- Data
Sweep It Under the Rug

Cover it up to keep other areas clean
(Façade and other Wrapper Patterns)
Put a rug at the Front Door

Protect Important Components!

Keep other parts of the system clean.

Sometimes Glue code (Mediators) helps keep others parts of the system cleaner.

(Anti-Corruption Layer -- Eric Evans)
Wipe your Feet at the Front Door

**ALIAS: ENCAPSULATE AND IGNORE**

**KEEPING THE INTERNALS CLEAN**

Patterns for Sustaining Architecture

PLoP 2012 Paper
Wipe your Feet at the Front Door

Sustaining Your Architecture

Filtering and Cleansing Sequence to keep Place Order Interface Clean

Protected Components

Adapter/Proxy/Façade
Frontdoor Wrapper
Filtering, Cleansing, Security Checks

Checks Files from Cust Site
Store Files to backup folder encrypted
Validates
Groups, Sorts and Filters
Generate XML for import
Place Order

S1
S2
S3
Sn
In my youth...two bad words

M and R words

Metadata and Reflection

Sustaining Your Architecture
Metadata
The Power of Metadata

Code is data, data is code. Everything is data. And data can drive 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"
Dynamic Hook Points

Eli Acherkan, Atzmon Hen-Tov, David H. Lorenz, Lior Schachter, Rebecca Wirfs-Brock, Joseph W. Yoder
Asian PLoP 2012

When building dynamic systems, it is often the case that new behavior is needed which is not supported by the core architecture. One way to vary the behavior quickly is to provide well-defined variation points, called hook-points, in predefined places in the systems where new behavior can be dynamically looked up and invoke at runtime when desired.
Dynamic Hook Points

Eli Acherkan, Atzmon Hen-Tov, David H. Lorenz, Lior Schachter, Rebecca Wirfs-Brock, Joseph W. Yoder
Asian PLoP 2012
Paving over the Wagon Trail

Patterns for Sustaining Architecture
PLoP 2012 Paper

ALIAS: MAKE REPETITIVE TASKS EASIER
STREAMLINING REPETITIVE CODE TASKS

Create simple examples, templates, & scripts
Develop a tool that generates code
Identify and use existing tools or frameworks
Develop a framework &/or runtime environment
Develop a domain-specific language
Paving over the Wagon Trail

Patterns for Sustaining Architecture
PLoP 2012 Paper
IMPORT EXAMPLE:

HOW TO DEAL WITH CLIENT VARIABILITY?
Import Orders

- Provided a standard format for imports
- Lots of duplication
- If Clients needed to vary from that format, we had to write or update a complete, self-contained custom import program
- Imports that need special processing, or had client specific rules, required to write a complete, self-contained import program
Sustaining Your Architecture

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.

Gather/Import/Reconciliation Process Flow

Note: In between all steps we’ll have hooks for overriding or adding additional behaviors or rules. Success and error info will be logged. Being able to reproduce bad imports is an important part of this. Therefore checkpoints will be added to each step for resubmitting at any point in the import process.
[ValidationRule("Joes Validation Rule")]

public class JoesValidationRule : ValidationRule {
    public JoesValidationRule () : base {
        public override void Validate(ImportContext context) {
            ...
        }
    }
}
Architectural Practice: Support Data Variability

- Separate what changes from stable part of system and provide support for adaptability/flexibility
- Integrate new learning into your architecture
  - Refactoring
  - Redesign
  - Rework
  - Code clean up
Sustaining an Architecture

- Minimize architectural debt: Support the ability to change/adapt what needs to change.
- Make what is too difficult, time consuming, or tedious easier.
- Decide at the most responsible moment, not the last possible moment.
- Learn and evolve.

Keep the system “livable” for its users and developers.
Don’t Pave the Cow Path

Don’t automate just because you can

Be careful how far you go!!!

Only add what variability you need when you need it at the most responsible time

Patterns for Sustaining Architecture
PLoP 2012 Paper
Don’t Pave the Cow Path
Patterns for Sustaining Architecture
PLoP 2012 Paper
Design Values
Supporting Variability

- Respect your system’s shearing layers
  - Understand the rates of what changes
- Determine & support who should be able to make changes, when, and at what cost
- Make what is too difficult, time consuming, or tedious easier
  - Create tools, leverage design patterns, use data to drive behavior…
- Don’t overdesign!!! (Only design what you need when you need it)

Sustaining Your Architecture
Resources

➢ Adaptive Object Models
  • www.adaptiveobjectmodel.com
  • www.metaplop.org
  • Contact us for training or onsite consulting

➢ Agile Software
  • Agile Alliance: www.agilealliance.org
  • The Agile Manifesto
  • 12 Principles of Agile Development

➢ Refactoring www.refactory.com
Thanks!!!

joe@refactory.com
Twitter: @metayoda