A model-driven software factory to support enterprise application product lines

Vinay Kulkarni
Tata Research Development and Design Centre, Pune, INDIA
vinay.vkulkarni@tcs.com
Outline

Motivation
Business applications
Our Model-driven approach
Our Model-driven factory
   A lightweight variant
Evaluation of factory approach
   Key characteristics
   Model-driven variant
   Lightweight variant
Experience and lessons learnt
Summary
Credits

All associated with development of *MasterCraft*

and

In memory of the one who is no more
Thank You
Software gets slower faster than hardware gets faster
- Wirth’s Law

The number of transistors on an integrated circuit for minimum component cost doubles every 24 months
- Moore’s Law

~30% projects get cancelled midway
~50% projects incur ~100% extra cost
~9% projects come on-time and on-budget
But, with only ~42% of planned features
- Standish Group report 1995

For every 25% increase in complexity of the tasks being automated, the complexity of the software solution itself increases by 100%
- Brooks’s Law

Adding manpower to a late software project makes it later
- Brooks’s Law
[The major cause of the software crisis is] that the machines have become several orders of magnitude more powerful! To put it quite bluntly: as long as there were no machines, programming was no problem at all; when we had a few weak computers, programming became a mild problem, and now we have gigantic computers, programming has become an equally gigantic problem.

– Edsger Dijkstra, *The Humble Programmer*

Computer science is no more about computers than astronomy is about telescopes.
Libraries
Frameworks
Application generators

Abstraction

Models
Predefined functionality with placeholders

Automation

Frameworks
Oft required computations

Automation

Application generators
Architecturally complete application

Process

Ad hoc
Waterfall
Iterative
Agile

Less *how* and more *what*

Abstract away the implementation platform

Abstract away the hardware

Predefined functionality with placeholders

Freedom from repetitive mundane tasks

Oft required computations

Architecturally complete application

When, Where and by Whom

When, Where and by Whom
Factory approach

Characterized by
- Tools replacing artisans for repetitive tasks
- A well-defined method
- Standards
- Flexible assembly-line

So as to deliver
- Higher productivity
- Uniformly high quality

in a *person independent* manner

*Can we apply the factory approach to software development?*
Enterprise applications

Business-critical
- High performance, Availability, Maintainability
  - Low algorithmic complexity, database-centric

Multitude of technologies
- Team of diverse skills

Long life
- Need to quickly respond to
  - Process change
  - Business change
  - Technology change

Large size
- Further complicates the development
Message round trip in a typical distributed architecture

Client
Form message
Send Message

Server
Receive Message
Process Message
Return Message

Client
Receive Message
Display
Interesting observations

Separation of concerns
– Architectural: GUI, Business logic, Database access etc
– Functional: Business logic Vs Solution architecture

Variety of patterns
– Database access
  • Key-based object access (CRUD)
  • Association-based object sets
– Presentation
  • Form
  • List as a result of Search
  • Aggregation of parts
Interesting observations

Variety of patterns

• Processing
  • Validate parameters – Raise Errors - Perform computations – Commit to database – Return

• Across architectural layers
  • Displaying large volume data in a ‘paged’ manner

Strategies

• Audit – *What? When? How?*
• Locks – Optimistic Vs Actual
• Deletes – Soft Vs Hard
• Exception handling context – *What? How?*
Interesting observations

- All patterns are data-centric
- Lends for easier specification
- Lends for framework definition
- Lends for automatic code generation for implementing the pattern
- Can be kept independent of implementation technology
Form Message on Client

Set functions and event handlers

Call `sh_mySvc(inOut, in1);`
Hand over to middleware

Status = sh_mySvc( OLTP buffer )
Security

Security Manager

Forward Txn after security validations

Logical Server

{ sh_mySvc1
  sh_mySvc2
  :
  :
  sh_mySvcn
}
Receive Message on Server

Logical Server

Sh_mySvc( oltp buffer )
Process Message on Server

• Construct message objects from OLTP buffer
• Invoke actual function
• On Success
  – Commit Transaction
  – Put return value(s) into OLTP buffer
• On Failure
  – Abort Transaction
  – Put error context into OLTP buffer
• Return to UI manager with status
Construct message objects from OLTP buffer

```
errorStatus = inOut->mySvc( in1 );
```
Server Side Processing

Output object of Query

inOut

Query Object

in1

Table

Create ()
Get ()
Modify ()
Delete ()
Exists ()

Get ()
Exists ()
Put return value(s) into OLTP buffer

Return errorStatus;
Process Return in UI manager

UI Manager

< myWindowHandle, sh_mySvc > ...

Client Memory

InOut::ClassId
InOut::specFlag
Attributes of InOut

OLTP Buffer
Display return value(s) onto Screen

Get functions and event handlers

OK Cancel
Example: GUI visualization and interaction

• Visualization
  – Form – used for displaying an object
  – List – used for displaying a related set of objects
  – Datatype specific GUI controls e.g. calendar control

• Interaction
  – Invocation of a service
  – Invocation of a child window
  – Paging behaviour for displaying large dataset in a window
Example: Handling of a set of objects

Service ( parameters, continuation key )
{
    : db_access_query_func( continuation key )
    :
}

Db_access_query_func ( continuation key )
{
    : continuation key contributes additional where clause to the original SQL statement
    :
}
Example: Object-relational map

- Inherently hierarchical world of objects needs to be supported by the inherently flat world of relational tables

- Different strategies:
  - Entire hierarchy in a single table
  - Each class maps to a separate table
  - Each class maps to a separate table but with replication of its parent’s attributes

- CRUD methods should reflect the chosen strategy
- Non-primary key based accesses should reflect the chosen strategy
Intuition

Generate solution architecture from its declarative specification
What is a model?

A **model** is a pattern, plan, representation, or description designed to show the structure or workings of an object, system, or concept.
What is a model?

A model is an abstract representation of some concrete information.
What is a meta model?

Structural correctness and consistency can be specified at meta model level
Model driven development - basics

- **Meta Meta Model**: A language to define all possible meta models
- **Meta Model**: A language for specifying a class of models
- **Model**: Abstract platform-independent specification
- **Tool**: Model2Model and Model2Text transformation
- **Code, Documents, Deployment Specs etc**: Text composition
Our model driven approach

1. Meta Meta Model
   - Instance of
2. Meta Model
   - Instance of
3. Model
   - Tool
4. Code, Documents, Deployment Specs etc
5. MOF
   - A MOF-describable meta model
6. A MOF-describable meta model instance
7. MOF Model to Text Transformation
8. Text composition
Model driven development - example

Meta Meta Model
  Instance of
  Meta Model
    Instance of
    Model

MOF
  GUI-layer Meta model
    GUI-layer Model
      Tool
        GUI-layer code

  App-layer Meta model
    App-layer Model
      Tool
        App-layer code

  DM-layer Meta model
    DM-layer Model
      Tool
        DM-layer code

Compose
Integrated implementation
Generating architectural code from models

class Employee
{
    // Attribute declarations
    String id;
    Double salary;

    // Attribute set/get/isSpecified methods
    void Setid(String pid)
    {
        id = pid;
    }
    String Getid() {return id;}
    void Setsalary(Double psalary)
    {
        salary = psalary;
    }
    Double Getsalary() {return salary;}
}

Oft-repeating code patterns can be automatically generated
Few more code patterns

Data Manager layer
- Persistence
  - How to map an inherently hierarchical world of objects onto inherently flat world of relational tables
- Optimistic locking
  - Counter or date-time-stamp
- Soft delete
- Caching

GUI layer
- Form
- Display results of a query
  - Entirely or in a paged manner
- Display data
  - Always use calendar control for date type and radio button for mandatory choice

A pattern is encapsulated in a meta model so as to generate the entire code thus delivering enhanced productivity and uniformly high quality
Shifting the focus away from coding

Why?

• Helps prevention and early detection of errors
• Can be automatically translated to code and other SDLC artefacts
• Can be retargeted to different technology platforms

How?

• Various code patterns recur in an implementation
• Capture a pattern using models and / or HLLs
• Implement a pattern-specific code generator to generate the corresponding code snippet
• Integrate code snippets in a consistent manner through an implementation architecture
Requirements of MDD toolset

- **Meta Model**
  - Instance of
  - **Model**
  - **Model’**
  - Xfm
  - Xfm
  - Code, Documents, Deployment Specs etc

- **Integration Bus**
  - **Meta modeling**
  - **Modeling**
  - **Model-to-Model Xfm**
  - **Model-to-Text Xfm**
  - **Model versioning**
  - **IDE**
Our MDD toolset

- Reflexive modeling tool
  - Meta modeling as well as Modeling tool
  - Extensible
  - Architected for distributed workspaces

- Code generators
  - Model-to-model and Model-to-text transformers
  - Extensible

- HLL
  - Simplified Business Oriented OO language
  - Translator(s) to popular OO language(s)

- IDE
  - Uniform Role-based interaction
Our model-driven factory

Key Enablers

**Reflexive modeling framework**
- Core meta meta model
- Constraint specification
- Extensible retargettable diagrammer

**Model-to-model transformation**

**Model-to-text transformation**

**Building block abstraction**

In operation

**Rapid functional prototyping**

**Architectural prototyping**

Setting up the factory
- Generating code generators
- Setting up process and IDE

Running the factory
- Using IDE

Customizing the factory

Benefits and pain points
Uniform N-layer modeling framework

Structuring concepts
- Workspace
- Packages
  - Nesting Semantics
  - Import Dependency Semantics
  - Type dependency Semantics
- Object
  - visibility (private / public)

Features
- Content based identity
- Workspace and VersionStore separation
- Storage transparency (In memory / repository ...)
- Class-specific storage extensions
- Class-specific UI extensions
- Delta driven processes
Core meta meta model

How is it related to MOF?
A sample meta model

Specifying constraints - OCL?
A sample rule and a script

/* Abstract class cannot be persistent and should not be mapped to a table */

Rule : 1 :Abstract_Class
{
    Class c; Table t;
    APPLYTO(c);
    IF (c.Abstract = 'A') THEN c.Persistent <> 'P' AND NOT_EXISTS (t | REL(c, t, 'MapsTo'));
    ISSUES(I0001, c);
}

Navigating model and producing text

FOREACH c IN Class
{
    // Print class <classname>
    : class $c.Name {
        // For each DataMember, associated to a Class ...
        FOREACH d IN c.HasData-DataMember
        {
            // print DataMember type and name
            : t\$d.Type $d.Name ;
        }
    }
}

How does it relate to OCL, MTT?
Extensible retargettable diagramming framework - conceptual overview

Diagram Interchange meta model

Application meta model

GUI controls meta model

Mapping

M

C

V
Diagram Types can be customized for different meta models
## A sample diagram specification

### Sample symbol specification:

```plaintext
SYMBOL CLASS, 1004, 180, 120
{
   DESCRIPTION "Class"
   SHAPE
   {
      RECTANGLE 0, 0, 180, 120 ;
      TEXT 1, 0, 0, 160, 120, "Class", 0
      :TX_HEIGHT=10
      :TX_FONT="Arial"
   }
   BOUNDARY
   {
      0, 0, 0, 120, 180, 120, 180, 0, 0, 0
   }
}
```

### Sample map specification:

```plaintext
symbol Class
{
   icon = "1004"
   MetaObject = Class
   symbolprop
   {
      "1" = MetaProperty(Name);
   }
}
```

---

**Rendering on different presentation kits?**
Why Model Transformation?

• Different models to specify different aspects of a system e.g.
  – Process/activity models to specify business processes
  – Class/ER models to specify business data
  – Use case and scenario models to specify interaction scenarios
  – State charts to specify control behavior

• But, only Class models for implementation

• Also, various design patterns need to be incorporated into models e.g.
  – Different object-relational mapping strategies, auditing etc
OMG QVT

- Specify mappings between models in terms of their meta models
- A mapping specification should enable:
  - Derivation of one model from the other
  - Checking consistency between models
  - Propagating changes from one model to the other - both additions and modifications (i.e, updates and deletes)
OMG Mof to Text

- Essentially, expressing the model content in a linearized text template which has a constant text part and placeholders for values to be obtained from the model.
- Ability to specify protected areas in the generated text that need to be preserved across successive applications of a MTT transformation.
- Emphasis not on ability to specify complex transformations as they are taken care of by QVT.
- Ability to modularize, compose, extend and reuse transformations.
Intuition

Model

Model-based code generator

Code

Repository of building blocks

Selection

Modification

Composition

Generic code generator generator
A building block encapsulates a choice along A, D and T dimensions

• Unifying abstraction to specify
  – [Concern] Model
  – Model transformation
  – Model translation

• Standards-based
  – OMG QVT
  – OMG MTT
A code generator is suitable composition of building blocks of choice
Building block types

• A leaf building block
  – Structure is defined in terms of parameterised meta model
  – Tool behaviour is specified using a meta model aware high level specification language (Mof to text)

• A non-leaf building block
  – Structure is defined as a composition of its constituent templates
  – Compositional behaviour denotes composition of the code corresponding to its constituent templates and is specified using a high level specification language (CSL)
Rapid customizability through interpretation

• Application functionality captured in the form of high level abstractions
  – Static behaviour
    • GUI: $\text{G}_{\text{models}}$, Business logic: $\text{F}_{\text{models}}$ and Database: $\text{D}_{\text{models}}$
  – Dynamic behaviour
    • Services
      – Business logic: Q++/EGL and Database access: SQL
    • Business process
      – BPEL

• Reflexive repository
  – Stores meta models, models and application objects
  – Provides a uniform API

• Interpreters query repository for type information and create application object instances therein (show in picture)
Architectural prototyping

- Prototypical functionality that covers all the architectural layers

Objectives
- Technology platform
- Design strategies
- How to best leverage underlying technology platform?
- Code patterns, coding guidelines, best practices etc

Crack team of technical architects

Hand-coded

Tested for representative workload

Iterate-till-saturate process

Normally, knowledge transfer from architects to developers precedes actual development

Factory provides abstraction to encapsulate these decisions in the form of building blocks which lead to definition of purpose-specific code generators

Operationalization of expertise
Flexible assembly-line

Tools to build tools to build …
A sample hierarchical composition

and a sample input model
Code generation

Instantiation step
– Models are stamped out (leaf level)
– Models are merged (non-leaf level)

Transformation step
– Code fragments are generated (leaf level)
– Weaving specifications are generated (non-leaf level)

Composition step
– Code fragments are weaved as per the weaving specifications

*Code generation process is a 3-step walk of the hierarchical composition*
After instantiation

- Person: Class
  - Ssn: Attribute
  - Name: Attribute
  - mapsTo
    - [O] Person: Table
      - audit
        - [A] Audit_Person: Table
          - [A] timeStamp: Column
          - [A] PreImage: Column
          - [A] PostImage: Column
  - [O] Person: Key

- [O] Create: Operation
- [O] Modify: Operation
- [O] Delete: Operation
- [O] Get: Operation
- [O] Exists: Operation
- [A] GetImage: Operation
- [A] SetPreImage: Operation
- [A] SetPostImage: Operation

[O] contribution of Object-Relational map template
[A] contribution of Audit template
After transformation

Package O-R-Map
Person::Modify()
{
    // Implementation
}
Person::Create()
Person::Delete()
Person::Get()
Person::Exists()

Package Audit
Person::GetImage()
{
    // Implementation
}
Person::SetPreImage()
Person::SetPostImage()

Bracket Person::Modify() Before GetImage(), SetPreImage()
After GetImage(), SetPostImage()
After composition

Package Persistence
Person::Modify()
{
    GetImage();
    SetPreImage();
    // Original body of Modify
    GetImage();
    SetPostImage();
}

Person::Create()
Person::Delete()
Person::Get()
Person::Exists()
Person::GetImage()
Person::SetPreImage()
Person::SetPostImage()
Meta model for IDE

- Process
  - Step
    - Manual
    - Automated
- Tool
  - Action
    - RoleView
- Role
  - Plays
    - Workspace
      - Component
    - User

Manual and Automated steps can be managed through Tool actions involving RoleView. RoleView roles, Workspace components, and User plays are defined.
Architecture

Model repository

Code generators

Message broker

Integration framework

Role based interaction
Customizing the factory

• Define meta models and HLLs to support new abstractions
• Define tools to process new models and HLLs
• Retarget existing tools to new technology platforms
• Integrate these newly created tools into the IDE
• Define new roles supported by IDE

Building block abstraction can address F and P dimensions as well
...customizing the factory

Feature model of a domain

Building Blocks

Generic Factory

Purpose-specific Selection

Mapping

Feature tree

BB

Customization

Specification Generator

Application Specifications

Customized Toolset

Code Generator Generator

Application implementation

Extension
Benefits and pain points

😊 Technology-proofing
😊 Increased productivity
😊 Uniformly high code quality
  ✓ Consistent implementation of design strategies, best practices, guidelines etc.
  ✓ Prevention and early detection of errors
  ✓ More complete testing of application

😊 Enhanced reusability
😊 Paucity of modeling experts
😊 Interoperability standards are emerging only now
😊 Customizing code generators takes time
😊 Less agile development process
  😞 Tight coupling with repository
  😞 Unidirectional synchronization from model to code
  😞 No debugging support at model level
  😞 Complex versioning and configuration management
Intuition

Generate solution architecture from its declarative specification
Code generation

- Class model
- Tag processor
- Enriched models
- Model to Text transformer
- Code fragments
- Weaver
- Business logic with metadata
- Tag definitions

Architect

Developer
Tag abstraction

Encapsulates non-functional characteristics
- Strategies: Persistence, Auditing, Archival, Logging, Error handling etc
- Platforms: RDBMSs, Presentation managers, EJB servers etc
- Architectures: JMI, EJB etc

Unifying abstraction to specify
- Data contribution i.e. attributes
- Method contribution i.e. code fragments
- Composition of code fragments

Specified using
- Extensible Java meta data
- Model-to-text transformation language

Enables
- Realization of a solution architecture as a composition of tags
- A repository of reusable tags
- J2EE development as a component factory

Tag is manifestation of building block abstraction in code
Operational view

Tool building
- Define solution architecture
- Identify tags to support realization of the solution architecture
- Convert code snippets into tag definitions
- Test tags for correctness w.r.t. solution architecture
- Release tags and their processor as a plugin

Tool usage
- Write business logic in Java
- Annotate appropriately with tags
- Generate complete class implementation
- Unit Test and release tested class
- System build + test
- Deploy
Extensibility

All services that are part of secure process steps need to be audited in a different manner.

Purpose-specific meta models can be defined
- Enhanced code generation
- Sophisticated weaving
Interoperable with model-centric approach

- Tag processor
- Class model
  - Tag Definition
  - Java Parser / JDT
- extended UML
  - Class models
- Model-based
  - Generators
  - Class Skeleton
  - Method and attribute contribution
- Translator
  - Q++/Query
- Composer
- Complete class implementation
  - with tag annotations
Benefits and pain points

😊 Uses industry standard technologies
😊 Quick turn-around time
😊 No runtime footprint
😊 No vendor lock-in
😊 Generated application can be maintained using only JDK
😊 Lightweight loosely coupled development process
😊 Quick development of purpose-specific tools
😊 Use of toolset can be easily switched off any time
😊 Leads to a repository of reusable software artefacts
😊 Limitations due to restricted meta model
Tools automating repeated tasks

Meta Model

Model

Model’

Code, Documents, Deployment Specs etc

Integration Bus

Meta modeling

Modeling

Model-to-Model Xfm

Model-to-Text Xfm

Model versioning

IDE
Well-defined method ensures repeatability

Analysis
- Class models, Use case models, Process models, Business rules, NFR specs
  - Identify business components
  - Refinement
  - Satisfy

Design
- Class models, DB models, GUI models, Identify design strategies
  - Identify to-be-deployed components

Construction
- Business logic in Q++ and SQL, GUI event code, Business rules in rules language

Unit testing
- Specify test cases
  - Test data and test harness generation
  - Test GUI
  - Test services

Prepare for deployment
- Package the generated system as per the specified deployment strategy

TATA CONSULTANCY SERVICES
2 February 2009
Standards

• Why standards?
  – Interoperable tools
  – Assembling a tool chain

• Modeling standards
  – A language to define desired models – MOF
  – A modeling language - UML
  – A language to specify model to model transformations - QVT
  – A language to specify model to text transformations - MTT
  – A text format to export/import model to/from – XMI

• Assembly line standards (for tools)
  – A process
  – A core bus to which every tool plugs in
Evaluation of model-driven factory

Of generated application
- **Evolvability** – Platform independent models + model based code generation
- **Maintainability** – Explicitly modeled dependencies + exact impact computation
- **Scalability** – Delegated to industry standard platforms
- **Extensibility** – Meta modeling
- **Integrability** – Layered architecture having well-defined interfaces

Of development (process)
- **Independence** – Component abstraction
- **Change isolation** – Separate workspaces
- **Fast turn-around time (for a change)** – Optimal impact computation + incremental code generation and build
- **Guarantee of integration** – Unified meta model
Evaluation of lightweight factory

Of generated application
- Maintainability – Separation of functional and architectural concerns
- Scalability – Delegated to industry standard platforms
- Extensibility – Meta modeling + Tag composition
- Integrability – Layered architecture having well-defined interfaces

Of development (process)
- Independence –
- Change isolation –
- Fast turn-around time (for a change) – Code-centric development process
- Guarantee of integration – Consistent Tag composition
Experience and lessons learnt

Over the past 14 years have delivered 70+ large business applications on a variety of technology platforms

Technical
• Increased productivity, uniformly high quality and platform independence
• Limited success on reuse front and that too in very recent past
• MOF and MTT sufficed majority of our common needs
• Need ready-to-use models of various NFRs

Managerial
• Have found it easy to convince product-owners and managers of large projects
• OMG standardization process has helped
• Large cycle-times for changes and steep learning curve are still the major hurdles
• Small projects still prefer code-centric development

Social
• MDD not yet established as a ‘domain’ – skills are still platform-specific 😇
• Needs greater involvement of academia
• Community focus needs to expand beyond ‘D’ to Analysis, Testing, Integration, Maintenance and Re-engineering
Summary

• Model-driven approach seems a right fit for business product-lines and large projects
• Code-centric approaches also can derive code generation benefits and interoperate with model-driven approach
• Availability infrastructure MDD technology and MDD standards enables a software factory
• Community should enhance the scope to cover A, T, I, M and R.
• Should also focus on creating ‘content’ in model form