Domain-Frontier approach to MDA based software development
Software Engineering History – Continual Raise of Abstraction
Software Project Development – Gradual Lowering of Abstraction
Software Engineering Evolution and Software Product Evolution Follow the Same Paths, but in Opposite Directions

Future

Creative Efforts

Manual Efforts

Scientific Facts

Automatic Compilers

Problem
Software Development  Yesterday, Today and Tomorrow

- Software Trends Today
  - Visual Programming Languages
  - Frameworks and Patterns
  - Pervasive Services Solutions
  - Partial Code Generators
  - Specific Domain Complete Generators, etc.
Software Development: Yesterday, Today, and Tomorrow

- MDA Unifies and Standardizes Modern Trends in Software Development
- Formalized System Analysis Method?
A Glimpse to the Future

- Formal Problem (Requirements) Definition Language?
- Will Domain Experts Be Able to Specify and Produce Software?
Domain-Frontier Approach Paradigms

- **MOF and UML – Unified Modeling Language**
  - Standard Notation
  - Standard Models Repository
  - Executable UML

- **Aspect Oriented Analysis and Design**
  - Overall Separation of Concerns
    - Problem Level: Domain and Frontier
    - System Level: Domain and Pervasive Services

- **Model Driven Architecture**
  - Model Centric Development
  - Model Transformations, etc.
Abstraction Management
- Clear Abstraction Milestones:
  - Problem → Architecture → Solution
  - Early Milestones without ambiguity

Dynamics at High Level Model
- Capture Behaviour from Begining

Focused on System External Interface
- Low Domain-Frontier Correlation
- User Interface Deserves Special Attention!
Domain-Frontier Models

- **Problem Model (“CIM”) – UML with OCL**
  - Business Process Model
  - Use Case Model
  - Conceptual Model with States
  - Sequence Diagrams
  - OCL Code

- **Architecture Model (“PIM”) – UML with Actions**
  - Analysis Model (Boundary, Control, Entity) with States
  - Sequence Diagrams
  - Action Semantics Code

- **Solution Model (“PSM”) – UML with Target Language**
  - Class Model
  - Component Model
  - Target Platform Code
Development Process Map (cont.)

Milestones:

- **Problem Modelling**
  - Requirements Understood
  - Architecture Generated

- **Architecture Tuning**
  - Design Generated
  - Scenarios Tested

- **Solution Tuning**
  - Solution Generated and Tested

- **Solution Implantation**
  - Solution Installed
Solution Tuning Phase

- Unit Testing
- Solution Refactoring
- Integration Testing
- System Testing
- Installation Package Configuration
- Installation Package Generation

Architect
Tester
Platform Specialist

Logger
System

Order Process
Breakfast Order

Order Process
Breakfast

«testcase»
Confirm Order

Domain
Logging Service

UI
Confirm Order

Order Process
Breakfast Order

System

Order Process
Logger

“Rosa has founded a company that supplies a complete breakfast delivered to the homes of its customers. Customers can order from one of the breakfast menus on Rosa’s Web site, indicate the hour and the place of delivery, give their credit card number, and the ordered breakfast will be delivered.

... 

Rosa wants to give to her customers a bit of flexibility. Customers may, after choosing a standard breakfast as basis, decide to put in extra comestibles, alter the amount of certain parts, and even remove parts from the breakfast.”

* Example problem statement taken, with permission, from “MDA Explained”, by Anneke Klepe, Jos Warmer and Wim Bast
The First Phase: Problem Modelling
Business Modelling

- Artifact: Business Process Model
- Semantics: Low to High

- Purpose: Understand Business Processes
- Variable Detail Level
- Integration with Workflow Engine?
- Early Class/State Identification
- Use Case Sequencing
- **Artifact:** Conceptual Model
- **Semantics:** High
- **Goal:** Understand Problem Structure
- Concepts Lifecycles
- Rosa Example: BreakfastOrder

- Breakfast Order:
  - Initial: EMPTY
  - Ready: READY
    - Ready to order: add(Breakfast)
    - Ready to cancel: cancel
  - Checking: CHECKING
    - Checking credit card: verify(Credit Card Nr)
      - Approved: CONFIRMED
        - Confirmed to dispatch: dispach
      - Rejected: CANCELED
        - Canceled: CANCELED
          - Canceled to remove: remove(Breakfast)
            - Breaks breakfasts = 1
          - Canceled to add: add(Breakfast)
            - Breaks breakfasts > 1
  - Closed: CLOSED
    - Closed to remove: remove(Breakfast)
      - Breakfasts = 1
    - Closed to add: add(Breakfast)
  - Dispatched: DISPACHED
    - Dispatched to close: close
Use Case Modelling

- **Artifact:** Use Case Model
- **Semantics:** Medium
- **Purpose:** Understand System Requirements
- State Machine per Use Case
- State per User Interface Presentation Unit (Page, Form, Electronic Device Screen, etc.)
System Dynamics Modelling

- Problem Level
  Interaction Diagram per Use Case
- All Paths Covered
- Domain Details Ignored
- Role: UI Designer

- Domain Level
  Interaction Diagram per Use Case
- Domain Internal Details
- Role: System Analyst
- **Unified Architecture Metamodel** (UML Analysis Model with States and Actions)

- **Tasks**
  - Select Architecture
    - Client/Server, Enterprise, Embedded, etc.
  - Mark Problem Model (Manual)
  - Generate Architecture Model (Automatic)

- **Model Transformations**
  - UML Metamodel(Problem) $\rightarrow$ UML Metamodel (Architecture)
    - Problem Structure $\rightarrow$ Architecture Structure
    - Problem Dynamics $\rightarrow$ Architecture Dynamics
  - OCL $\rightarrow$ Action Language
Rosa Example
  - Architecture: Enterprise

Transformations:
  - Business Model $\rightarrow$ Architecture Model
    (Enterprise Business Services Layer)
  - Use Case Model $\rightarrow$ Architecture Model
    (Enterprise Presentation Layer)
  - Domain Model $\rightarrow$ Architecture Model
    (Enterprise Business Components Layer)
  - Problem Interaction Models $\rightarrow$ Architecture Interaction Models
  - OCL $\rightarrow$ Action Language Instructions
### Business Model → Analysis Model

<table>
<thead>
<tr>
<th>Source</th>
<th>Destination</th>
</tr>
</thead>
<tbody>
<tr>
<td>Element</td>
<td>Stereotype</td>
</tr>
<tr>
<td>Activity</td>
<td>Process</td>
</tr>
<tr>
<td></td>
<td>Is Aggregated</td>
</tr>
<tr>
<td>Add. Cond.</td>
<td>Class</td>
</tr>
<tr>
<td>Add. Instr.</td>
<td>Control</td>
</tr>
<tr>
<td>Aggregate -</td>
<td>Aggregated</td>
</tr>
</tbody>
</table>

### Use Case Model → Analysis Model

<table>
<thead>
<tr>
<th>Source</th>
<th>Destination</th>
</tr>
</thead>
<tbody>
<tr>
<td>Element</td>
<td>Stereotype</td>
</tr>
<tr>
<td>State</td>
<td>Context UC is UI</td>
</tr>
<tr>
<td></td>
<td>Class</td>
</tr>
<tr>
<td></td>
<td>Boundary</td>
</tr>
<tr>
<td></td>
<td>isUI = true</td>
</tr>
<tr>
<td>State Transition</td>
<td>Message is Self</td>
</tr>
<tr>
<td></td>
<td>Association</td>
</tr>
<tr>
<td></td>
<td>Navigate</td>
</tr>
<tr>
<td></td>
<td>Start State Class</td>
</tr>
<tr>
<td></td>
<td>− End State Class</td>
</tr>
<tr>
<td>State Transition</td>
<td>Message is not Self</td>
</tr>
<tr>
<td></td>
<td>State Class</td>
</tr>
<tr>
<td></td>
<td>− Process Control Class</td>
</tr>
</tbody>
</table>
OCL → Action Language Compiler
Rosa Example: CalculatePrice() Operation

```plaintext
{context BreakfastOrder::calculatePrice()}
body:
sum( self.breakfasts->collect( comestible.price * Change.quantity ) )
```

```plaintext
select many breakfasts related by self->Breakfast[R7]
result = 0
for each breakfast in breakfasts
  select many comestibles related by breakfast->Comestible[R8]
  for each comestible in comestibles
    select one charge related by comestible->Charge[R8]
    result = result + comestible.price * charge.quantity
  next
next
```
The Second Phase: Architecture Tuning
Action Coding

- Complete State Procedures and Operations Code

Customer (from Use Case Model)

- Credit Card Input
- Credit Card Verification
- Credit Card Approved

Orders Process

- +order
- +breakfastOrder

Transactor

BreakfastOrder

Credit Card Company (from Use Case Model)
**Scenario Testing**

- Architecture Interaction Diagrams
  Generated from Problem Interaction Diagrams
- Architecture Model Virtual Machine
- Simulations – Execution Traces
Pervasive Services Identification and Configuration

- Identify Needed Pervasive Services (PS)
  - Select From Available PS Modules, or
  - Develop New PS Modules

- Rosa Example: Logging Service

```
<table>
<thead>
<tr>
<th>Operation</th>
<th>+status</th>
<th>&lt;-enumeration&gt; Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>name: String</td>
<td>0..*</td>
<td>- ok:</td>
</tr>
<tr>
<td>time: Date</td>
<td>1</td>
<td>- error:</td>
</tr>
<tr>
<td>result: String</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter</th>
<th>- name: String - Value:</th>
</tr>
</thead>
</table>

init(name, parameters)

log(status, result)

self.time = Now
self.status = rcvd_evt.status
self.status = rcvd_evt.result
```
Integrate System with Pervasive Services

Tasks:
- Mark the “Join Points” in the Architecture Model
- Weave the PS Modules with Architecture Model

Rosa Example:
- Requirement:
  All the Executions of theOrderProcess::Verify() should be registered
- Solution:
  Tag the OrderProcess::Verify() with “isLogged = true”

Mapping Rules are Part of Every Individual PS Module
OrderProcess::Verify( CreditCardNr: Integer, BreakfastOrderID: Integer): Boolean

//----- inserted by the Logger PS Weaver
create object instance operation of Operation
relate relate operation to self across R6
generate init("OrderProcess::Verify", {"CCNr", CreditCardNr, "OrderID", BreakfastOrderID}) to operation
//----- finished

// here goes the Verify() domain code

//----- inserted by the Logger PS Weaver
generate log(status, result) to operation
unrelate operation from self across R6
//----- finished
Solution Generation

- **Unified Architecture Metamodel** → Target Platform

  - Tasks:
    - Select the Platform (J2EE, .NET, etc)
    - Mark the Architecture Model (Manual)
    - Generate the Solution Model (Automatic)

- Model Transformations:
  - UML Metamodel (Architecture) → UML Metamodel (Platform)
  - Action Language → Target Platform Language
### Analysis Model → Solution Model

<table>
<thead>
<tr>
<th>Source</th>
<th>Destination</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Element</strong></td>
<td><strong>Stereotype</strong></td>
</tr>
<tr>
<td>Class</td>
<td>Boundary</td>
</tr>
<tr>
<td>Class</td>
<td>Boundary</td>
</tr>
<tr>
<td>Class</td>
<td>Boundary</td>
</tr>
<tr>
<td></td>
<td>Interface</td>
</tr>
<tr>
<td>Class</td>
<td>Control</td>
</tr>
<tr>
<td></td>
<td>Interface</td>
</tr>
<tr>
<td>Class</td>
<td>Entity</td>
</tr>
<tr>
<td></td>
<td>Interface</td>
</tr>
</tbody>
</table>
The Third Phase: Solution Tuning
- Target Platform Testing
  - Functional and no Functional
  - Unit and Integration
- Refactoring
- Platform Language Coding (if needed)
- Installation Package Generation
The Last Phase: Solution Implantation
Future Work

- User Interface Interaction Modelling Metamodel
  - UML Profile v.s. Alternative Metamodel

- Transformations
  - Transformation Definition Metamodel
  - Model Compiler v.s. Archetypes

- Architecture Model Infrastructure
  - Virtual Machine
  - Simulator

- Pervasive Services Integration
  - Aspect Oriented Technics Formalization
Bibliography

- “MDA Distilled, Principles of Model-Driven Architecture”, S. J. Mellor, K. Scott, A. Uhl, D. Weise
- “Model Driven Architecture, Applying MDA to Enterprise Computing”, D. S. Frankel
- “Mastering AspectJ”, J. D. Gradecki, N. Lesiecki
Thank You!

Aleksandar Orlic
aorlic@craftware.net