MDA Distilled

Stephen J. Mellor
Chief Scientist
Mentor Graphics Corp
Table of Contents

1. What’s the problem?
2. Models
3. Metamodels
4. Mappings
5. Marks
6. Building a Language
7. Agile MDA
8. Conclusion
Table of Contents

1. What’s the problem?
2. Models
3. Metamodels
4. Mappings
5. Marks
6. Building a Language
7. Agile MDA
8. Conclusion
What’s the Problem?

- Software is expensive, and productivity is low for many reasons:
  - Code is at too low level of abstraction
  - Reuse occurs (to the extent it does at all) at too low a granularity
  - Any code is glued together (at great expense) to its infrastructure (also expressed as code)
  - Mapping information (design expertise) is applied—then lost
The Evolution of Software Development

Price Performance

System Complexity

Assembly

Assembler

Machine Code

1980s

High Level Language

Compiler

Assembly

1990s

UML Models

Model Compiler

High Level Language

2000s

Increased Productivity

Accelerated Technology
Reuse Granularity

Components and frameworks require common infrastructure
Code Binds

- Code is glued to its infrastructure:
  - Binds device control to the database
  - Binds the copier to
    (device control and the database)
  - Binds the image to the
    (copier and
    (device control and
    the database))...
Mapping Information is Lost

- Mapping between layers is all skilled manual labor.
- And once a mapping is ‘found,’ it is applied by hand.
- When a change is made, the mappings are not repeatable.

All manual work!
Components of an MDA Solution

- Capture *each layer* in a platform-independent manner as intellectual property.

- Capture *the mappings* to the implementation as intellectual property (IP).

- *Models and mappings* become assets.
Enter Model-Driven Architecture

- MDA: an interoperability standard for combining models at design-time.

This enables a market for IP in software.
Enter Model-Driven Architecture

- **MDA:**
  - Captures IP as models and enables protection of them
  - Allows IP to be mapped automatically
  - Allows multiple implementations
  - Makes IP portable

*This enables a market for IP in software.*
Modeling Language for Software

“The Unified Modeling Language is a language for specifying, constructing, visualizing, and documenting the artifacts of a software-intensive system.” The UML Summary

© Object Management Group
Why Model?

- A good model:
  - Abstracts away not-currently-relevant stuff
  - Accurately reflects the relevant stuff, so it…
  - Helps us reason about our problem
  - Is cheaper to build than code
  - Communicates with people
  - Communicates with machines
What is a Model?

- A model is a coherent set of elements that:
  - Covers some subject matters
    - Doesn’t have to cover all subject matters
  - At some level of abstraction
    - Doesn’t have to define realizations
  - That need not expose everything
    - Doesn’t have to show everything at once
  - That need not be complete in itself
    - Doesn’t have to include “code”

- Seating plan?
- Materials?
- Interior?
- No engine yet!
Subject Matters

- Good models come from separating layers by subject matter, so that each one is *platform independent*.

- A change to models in one subject matter should not necessitate reconstruction of models in another subject matter.
Start with an abstract problem (e.g. a Bank), with an abstract modeling language (e.g. UML).

End with a concrete statement of the solution in a low-level concrete language (e.g. Java).
Abstraction and classification

Real World

- slug
- stray
- feral

Models

Classify

Fido(20Kg, Awful): Dog
Munchin(16Kg, FeedingOnly): Cat
LapKitty(12Kg, LapLover): Cat

Entity classifications

Class Model

Abstract

Pet
+ name
+ weight

Dog
+ slobberFactor

Cat
+ standOffIndex
Model Views

A diagram is a coherent view on a model.
Incompleteness

Code can be added to a model later.
Executable UML models

- UML can be used as a semantic modeling language, if we:
  - Define actions
  - Define the context
  - Define execution rules

For an underlying semantic model

- The underlying semantic model is an:
  - (x) executable
  - (t) translatable

UML
Defining Behavior Using UML

- UML can now be used to define behavior
  - UML 1.5/2.0 now has Action Semantics

- Use an executable translatable profile of UML ($X_T\text{UML}$)

- $X_T\text{UML}$ defines behavior without making premature design decisions
Three Primary Diagrams

- **Class Diagram**
- **Statechart Diagram**
- **Action Language**

Defines Lifecycle for Shutter

**Shutter**
- Shutter ID (I)
- Aperture
- Zoom
- OpenTime
- Status

**Exposure**
- Exposure # (I)
- Shutter ID (R4)
- NumberOfBytes
- FileFolder (R5)
- Status

**Checking Settings**
- Closed
- Half (ShutterID)
- Released (ShutterID)
- Checking Settings
- Full (ShutterID)
- Open
- Released (Shutter ID)

Action Language:

- **Entry**: `OpenShutter(0.5);`
- **MeasureLight();**
- **DetermineExposureTime(Mode);**
Table of Contents

1. What’s the problem?
2. Models
3. Metamodels
4. Mappings
5. Marks
6. Building a Language
7. Agile MDA
8. Conclusion
What is a Metamodel?

A metamodel captures developer models in a model repository
UML Metamodel
Instance-of

Real World

- slug
- stray
- feral

Models

- Abstract
  - Fido(20Kg, Awful): Dog
  - Munchin(16Kg, FeedingOnly): Cat
  - LapKitty(12Kg, LapLover): Cat

Real Entities

- Fido(20Kg, Awful): Dog
- Munchin(16Kg, FeedingOnly): Cat
- LapKitty(12Kg, LapLover): Cat

Instance Model

- Abstract
  - Pet
    - + name
    - + weight
  - Dog
    - + slobberFactor
  - Cat
    - + standOffIndex

Entity Classifications

- slug
- stray
- feral

Class Model

Real World

- Fido(20Kg, Awful): Dog
- Munchin(16Kg, FeedingOnly): Cat
- LapKitty(12Kg, LapLover): Cat

Abstract

- Pet
  - + name
  - + weight

Class Model

- Dog
  - + slobberFactor
- Cat
  - + standOffIndex
The Relationship to the Metamodel

Problem domain: A model

Entity classifications

Classify

Problem domain: A modeling language (i.e. a Metamodel)

Class Model

Instance of

Class

Attribute

Abstract

Pet
+ name
+ weight

Dog
+ slobberFactor

Cat
+ standOffIndex

Abstract

Classify

Problem domain: A model

Entity classifications

slug
stray
feral

Abstract

Class

Attribute

Dog
+ slobberFactor

Cat
+ standOffIndex

Abstract

Classify

Problem domain: A modeling language (i.e. a Metamodel)
Metamodel Instances

Just like an application model, the meta-model has instances

<table>
<thead>
<tr>
<th>Class ID</th>
<th>Name</th>
<th>Descr'n</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>Recipe</td>
<td>.....</td>
</tr>
<tr>
<td>101</td>
<td>Batch</td>
<td>.....</td>
</tr>
<tr>
<td>102</td>
<td>Temp Ramp</td>
<td>.....</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>State</th>
<th>Class ID</th>
<th>State #</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Filling</td>
<td>101</td>
<td>1</td>
<td>Filling</td>
</tr>
<tr>
<td>Cooking</td>
<td>101</td>
<td>2</td>
<td>Cooking</td>
</tr>
<tr>
<td>Emptying</td>
<td>101</td>
<td>3</td>
<td>Emptying</td>
</tr>
<tr>
<td>Emptying</td>
<td>102</td>
<td>1</td>
<td>....</td>
</tr>
<tr>
<td></td>
<td>102</td>
<td>2</td>
<td>.....</td>
</tr>
<tr>
<td></td>
<td>102</td>
<td>.....</td>
<td>.....</td>
</tr>
</tbody>
</table>

- Create Batch( Amount of Batch, Recipe Name)
- Filling
- Filled( Batch ID )
- Cooking
- Temperature Ramp Complete( Batch ID )
- Emptying
- Emptied( Batch ID )
### Recipe

<table>
<thead>
<tr>
<th>Recipe Name</th>
<th>Cooking Time</th>
<th>Cooking Temp</th>
<th>Heating Rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nylon</td>
<td>23</td>
<td>200</td>
<td>2.23</td>
</tr>
<tr>
<td>Kevlar</td>
<td>15</td>
<td>250</td>
<td>4.69</td>
</tr>
<tr>
<td>Stuff</td>
<td>45</td>
<td>280</td>
<td>1.82</td>
</tr>
</tbody>
</table>

### Batch

<table>
<thead>
<tr>
<th>Batch ID</th>
<th>Amount of Batch</th>
<th>Recipe Name</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>100</td>
<td>Nylon</td>
<td>Filling</td>
</tr>
<tr>
<td>2</td>
<td>127</td>
<td>Kevlar</td>
<td>Emptying</td>
</tr>
<tr>
<td>3</td>
<td>93</td>
<td>Nylon</td>
<td>Filling</td>
</tr>
<tr>
<td>4</td>
<td>123</td>
<td>Stuff</td>
<td>Cooking</td>
</tr>
</tbody>
</table>

### Model Instances (M0)

### MetaModel Schema (M1)

### MetaModel Instances (M1)

### Model Schema (M1)

### Model Instances (M0)
The “four-layer architecture” is a simple way to refer to each layer.

In reality, meta-levels are relative.
Fourth Layer

- The fourth layer is a *model of the metamodel*, which yields a “meta-meta-model.” It is the simplest model that can model the metamodel.

- A metamodel of the “meta-meta-model” (i.e. the “meta-meta-meta-model”) would have the same structure as the meta-meta-model. This layer is:
  - Meta-circular
  - Normally associated with the MOF

Meta? Did you say “meta?!”
MOF

- The **Meta-Object Facility** is an OMG standard that defines the structures for M3.

- *Any* metamodel can be captured in MOF (not just UML), which makes it the basis
  - for defining standards that ...
  - …map between metamodels.
Table of Contents

1. What’s the problem?
2. Models
3. Metamodels
4. Mappings
5. Marks
6. Building a Language
7. Agile MDA
8. Conclusion
Mapping Functions
A mapping function transforms one model into another
Types of mappings
In general, a mapping can be...

Abstract  Refining  Abstracting  Representing  Merging
Abstract  Language  Concrete

Subject Matter
Example of Merging Mapping

- Floor selection
- Cabin dispatching
- Door open/close timing

- Safe acceleration
- Precise transport

Elevator uses Transport Bridge between domains

Elevator

- Bank
- Shaft
- Cabin
- Door

Transport

- Load
- Axis of Motion
- Motor
- Acceleration Profile

Elevator uses Transport Bridge between domains

gotoFloor (Cabin 3, Floor 6)
cabinArrived ()
move (Load 14, Position 334.25, Ramp 3B)
moveCompleted ()
Defining Mapping Rules

- A metamodel allows us to state mapping rules
  - For each Class….
  - For each Structural Feature…
  - For each Attribute…
  - For each Action

- Rather than manipulate specific classes in the developer model.

- This means a standard “mapping tool” can be defined
Metamodel-Metamodel Mapping

- All models are manipulated through the MOF (Meta-Object Facility)

- Query
- View
- Transform

Elevator

Transport Infrastructure

Device I/O Infrastructure

User Interface Infrastructure

Underlying repository (MOF)
QVT

- There is presently no standard, but three approaches present themselves
  - Imperative
  - Template
  - Declarative
- The RFP explicitly demands declarative, but alternatives have been proposed
Metamodel-Metamodel Mappings

QVT is a standard approach for defining *mapping functions* that map between metamodels.

Inserts element ("attribute") in target metamodel.
Table of Contents

1. What’s the problem?
2. Models
3. Metamodels
4. Mappings
5. Marks
6. Building a Language
7. Agile MDA
8. Conclusion
Why Marks?

A *mark* distinguishes multiple possible targets
Applying Marks

- Marks may be used as
  - Rule selection
    - If the mark has value isRemote, invoke a remote accessor, otherwise...
  - Value provider
    - Prefix the (string) value to all marked elements (E.g. add the string “db_” to all db accessors)
Marking Models

- A *marking model* is a way to declare
  - Names of marks
  - Their types
  - Defaults (if any)

Invocation: 
```
Accessibility ::= [ isRemote | isLocal ] = isRemote
```

Accessor: `Name_Prefix ::= string`

```
Name
```

```
Type
```

```
Default
```

```
Name
```

```
Type
```
Relating Marks to Metamodel Types

- Marks are associated with metamodel elements

Invocation: Accessibility ::= [ isRemote | isLocal ] = isRemote

Accessor: Name_Prefix ::= “db_” : string
“Are Associated With?”

- Both *Invocation* and *Accessor* are UML metamodel elements.

- The marks *Accessibility* and *Name_prefix* describe these metamodel elements, but are *not* a part of them.
Other Marks

- Some marks are “constants”
  - For example, a postfix to all class names
- You can think of these as marks that apply to the metamodel (M2)
- Some marks apply to instances
  - For example, processor allocation for fixed-input devices
- You can think of these as marks that apply to the instance model (M1)
Theory of Marks

- There isn’t one. Yet.

- But:
  - What should be parameterized as a function vs. a mark?
  - Can there be a taxonomy of marks?
  - What are good/bad ways to use marks?
  - Should marks be prescriptive, or should they describe the source model and let the mapping function decide?
    - For example, is it better to say “linked list” or say “few instances,” which *might* then imply a linked list?
## Table of Contents

1. What’s the problem?
2. Models
3. Metamodels
4. Mappings
5. Marks
6. Building a Language
7. Agile MDA
8. Conclusion
Languages

- We build languages all the time
  - When we subset the UML for our preferred elements
  - When we extend it by adding adornments or notes
- We must decide if we need to do so formally
- Language definers include
  - Standards bodies
  - Tool vendors
  - Methodology definers
  - MDA architects
  - Developers
Building a Language Using MOF

- MOF is an (object-oriented) metamodeling language
  - It can be used to create a language
    - For example, UML
- You can use MOF to create your own modeling language
Building a Language Using Profiles

- A profile is a UML mechanism used to define and extend metamodels
  - Profiles may be used to define metamodels for PIMs and PSMs
  - Profiles may be used to define marking models

- A profile is defined in terms of
  - Stereotypes that extend “meta-”classes, and
  - Constraints, defined using OCL
Example

Figure 12-99: A simple EJB profile
Superstructure submission
Building Graphical Notation
(for a language)

- In a networking problem, we may want to draw:

- which may be captured as:

- with instances:
  - 4711
  - 42
  - 3, 4711
  - 7, 42
  - A, 3, 7, 173
Building Graphical Notation (for a language)

- By mapping the model to UML, we get drawing tools “free”
Table of Contents

1. What’s the problem?
2. Models
3. Metamodels
4. Mappings
5. Marks
6. Building a Language
7. Agile MDA
8. Conclusion
Elaborative Development

Analysis

Preliminary Design

Detailed Design

Target Code

Design and Architecture Details

Implementation Details and Code Bodies

Code Generation

Round Trip

Intermixed Application and Design

Manually Created Code Bodies and Implementation Details Required for Model Execution and Code Generation

Target Code assembled from Hand-Coded Bodies inserted into a generated framework
What’s Wrong With That?

- Each meta-model demands its own profile
- Each transformation goes through the MOF, but
  - *the transformations must be specific to the profile*
  - *even though the transformation language is standardized*
What’s the Solution?

- Model each domain using
  - A single neutral formalism that
  - (perforce) conforms to the same metamodel
What’s the Solution?

- Connect up the models according to
  - a single set of mapping rules that
  - operate on to the same metamodel
Metamodel-to-text Mappings

- MDA needs a way to map data from a metamodel into text.

```c
.function ClassDef( inst_ref class )
    class ${class.name} : public ActiveInstance {
        .private:
            .invoke PrivateDataMember( class )
    }
    ... 
.end function

.function PrivateDataMember()
    .select many PDMs related by class->attribute[R105]
    .for each PDM in PDMs
        ${PDM Type} ${PDM Name};
    .endfor
.end function
```

We call them "archetypes".
Example

- The archetype language produces text

```cpp
.select many states related to instances of class -> [R13] StateChart -> [R14] State
  where (selected.isFinal == FALSE) public:
    enum states_e
      { NO_STATE = 0 ,
    .for each state in states
      .if ( not last
        ${state.Name} ,
    .else
      NUM_STATES = ${state.Name}
    .endif
    .endfor
};

public:
    enum states_e
      { NO_STATE = 0 ,
        Filling ,
        Cooking ,
        NUM_STATES = Emptying
    }
```
Agile MDA

- Each model we build covers a single subject matter
- We use the same *executable* modeling language for all subject matters
- The executable model does not imply an implementation
- Compose the models automatically
- This last is *design-time composability—a bus*
Model Compilers

- A model compiler compiles each model according to a single set of architectural rules so that the various subject matters are known to fit together.

A model compiler
- Normalizes models to the infrastructure
- Combines models at design time
Model Compilers

- System dimensions include:
  - Concurrency and sequentialization
  - Multi-processing & multi-tasking
  - Persistence
  - Data structure choices
  - Data organization choices

= model compiler
(that executes the mappings)
### Examples

<table>
<thead>
<tr>
<th>Financial system</th>
<th>Embedded system</th>
</tr>
</thead>
<tbody>
<tr>
<td>Highly distributed</td>
<td>Single task</td>
</tr>
<tr>
<td>Concurrent</td>
<td></td>
</tr>
<tr>
<td>Transaction-safe with rollback</td>
<td>No operating system</td>
</tr>
<tr>
<td>Persistence, with rollback</td>
<td>Optimized data access and storage</td>
</tr>
<tr>
<td>C++</td>
<td>C</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Telecommunication system</th>
<th>Simulation system</th>
</tr>
</thead>
<tbody>
<tr>
<td>Highly distributed</td>
<td>Mostly synchronous</td>
</tr>
<tr>
<td>Asynchronous</td>
<td>Few tasks</td>
</tr>
<tr>
<td>Limited persistence capability</td>
<td>Special-purpose language: “Import”</td>
</tr>
<tr>
<td>C++</td>
<td></td>
</tr>
</tbody>
</table>
Building the System

**Application Models**

**Underlying Semantics Repository (MOF)**

<table>
<thead>
<tr>
<th>Class ID</th>
<th>Name</th>
<th>Desc'n</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>Recipe</td>
<td>....</td>
</tr>
<tr>
<td>101</td>
<td>Batch</td>
<td>Ramp</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Class ID</th>
<th>State #</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>1</td>
<td>Filling</td>
</tr>
<tr>
<td>101</td>
<td>2</td>
<td>Cooking</td>
</tr>
<tr>
<td>101</td>
<td>3</td>
<td>Emptying</td>
</tr>
<tr>
<td>102</td>
<td>1</td>
<td>....</td>
</tr>
<tr>
<td>102</td>
<td>2</td>
<td>....</td>
</tr>
</tbody>
</table>

**Libraries, Legacy or Hand-written code**

**Model Compiler**

**Archetypes**

**Execution Engine**

**Generator**

**Code for the System**

**Division of Mentor Graphics - Company Confidential - 2004**
All Domains are Translated

MDA models can have multiple implementations depending on the target environment.

Design is specific to category of platforms
# Table of Contents

1. What’s the Problem?  
2. Models  
3. Metamodels  
4. Mappings  
5. Marks  
6. Building a Language  
7. Agile MDA  
8. Conclusion
Building a Market

- Design time composability
  - protects IP
  - allows IP to be mapped to multiple implementations
  - enables a market in IP in software
MDA enables a market for IP in software!

- Code-driven development produces expenses.
- Model-driven development produces assets.
OMG TLAs

- **MOF** = Meta-Object Facility
  - a repository for metamodels
- **CWM** = Common Warehouse Metamodel
  - Can map between models
- **QVT** = Query/View/Transform,
  - A standard for mapping between (MOF) metamodels
  - Presently an RFP and not yet a standard
- **XMI** = XML Model Interchange
MDA Standardization

- UML 2.0 Infrastructure Sept 2004
- QVT (metamodel-metamodel) Nov 2004
- Marks Not Understood
- Action Language Necessary?
- Archetypes (metamodel-text) Issued

- The ADTF and the MDA WG proposes these RFPs.
Learn More

Comprehensive language introduction and reference

A complete set of models on CD

Practical guide for model developers

www.omg.org
www.projtech.com