MDA Distilled

Stephen J. Mellor
Vice-President
Project Technology, Inc.
http://www.projtech.com
Table of contents

1. What’s the problem?
2. Models
3. Metamodels
4. Mappings
5. Marks
6. Representing models
7. Agile MDA
8. Conclusion
What’s the problem?

Software is expensive, and productivity is low for many reasons. Amongst them:

- 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

No wonder!

Expensive and hard-to-find!
Language abstraction

High-level language source code is two-dimensional.

- **Assembly Code**: 1960’s
- **High Level Language Source Code**: 1980’s
- **Executable Models**: 2000’s

**Assembler**

**Source Code Compiler**

**Model Compiler**

**Sequential**: 1-D

**Block-structured**: 2-D

**Graphical**: 3-D
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 mappings is ‘found,’ it is applied by hand
- When a change is made, the mappings are not repeatable.
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.

© OMG
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.

© OMG
## Table of contents

1. What’s the problem?
2. Models
3. Metamodels
4. Mappings
5. Marks
6. Representing models
7. Agile MDA
8. Conclusion
“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
Abstraction and classification

Problem domain vs. Model

Abstract: name, weight, standOffIndex

Classify:
- Dog: + slobberFactor
- Cat: + standOffIndex

Types:
- Pet: + name + weight

Examples:
- Munchkin
- Fido
- LapKitty
- stray
- slug
- feral
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 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”

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).
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:

- executable
- 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

**Batch**
- Batch ID {I}
- Amount of Batch
- Recipe Name {R2}
- Status

**Temperature Ramp**
- Ramp ID {I}
- Batch ID {R4}
- Start Temperature
- Start Time
- End Temperature
- End Time
- Status

**Lifecycle for Temperature Ramp**

**Action for Creating**
- Creating
  - Entry/Create TemperatureRamp with BatchID, EndTime, EndTemp
  - Assign CurrentTime to Self.StartTemp;
  - ActualTemp to Self.StartTemp;
  - Generate StartControlling (Ramp ID );
Table of contents

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

A metamodel captures developer models in a model repository.

What is the structure of the repository?
UML metamodel
Abstract

Instance-of

Reflects

Fido(20Kg, Awful):Dog

Munchin(16Kg, FeedingOnly):Cat

LapKitty(12Kg, LapLover):Cat

Instances

Types

Problem domain

Classify

Instance of

Model

Pet
+ name
+ weight

Dog
+ slobberFactor

Cat
+ standOffIndex

slug

stray

feral

name, weight, standOffIndex

name, weight, slobberfactor

name, weight, standOffIndex
The relationship to the metamodel

Problem domain: Pets

- Pet
  - name
  - weight
  - standOffIndex

- Dog
  - name
  - weight
  - slobberFactor

- Cat
  - name
  - weight
  - standOffIndex

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

- Pet
  - name
  - weight

- Dog
  - slobberFactor

- Cat
  - standOffIndex

- Class

- Attribute
Metamodel instances

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

### Class

<table>
<thead>
<tr>
<th>Class ID</th>
<th>Name</th>
<th>Desc'r'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>

### State

<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>
<tr>
<td>102</td>
<td>.....</td>
<td>.....</td>
</tr>
</tbody>
</table>
Four-layer architecture

The “four-layer architecture” is a simple way to refer to each layer.

(In reality, meta-levels are relative.)
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:

- Reflective
- Normally associated with the 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. Representing models
7. Agile MDA
8. Conclusion
Mapping functions

A mapping function transforms one model into another.
Types of mappings

In general, a mapping can be:

- Refining
- Abstracting
- Representing
- Migrating
Example of merging mapping

- Floor selection
- Cabin dispatching
- Door open/close timing
- Safe acceleration
- Precise transport

Elevator uses Transport Bridge between domains

gotoFloor (Cabin 3, Floor 6)
cabinArrived ()
move (Load 14, Position 334.25, Ramp 3B)
moveCompleted ()
Metamodel-metamodel mappings

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

A metamodel (as stored in MOF) 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: QVT.
QVT is a standard approach for defining *mapping functions* that map between metamodels.

Inserts element ("attribute") in target metamodel.

- Query
- View
- Transform
There is presently no standard, but three approaches present themselves:

- Imperative,
- Template,
- Declarative.

The RFP explicitly demands declarative, but alternatives have been proposed.
<table>
<thead>
<tr>
<th>Chapter</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. What’s the problem?</td>
</tr>
<tr>
<td>2. Models</td>
</tr>
<tr>
<td>3. Metamodels</td>
</tr>
<tr>
<td>4. Mappings</td>
</tr>
<tr>
<td>5. Marks</td>
</tr>
<tr>
<td>6. Representing models</td>
</tr>
<tr>
<td>7. Agile MDA</td>
</tr>
<tr>
<td>8. Conclusion</td>
</tr>
</tbody>
</table>
Why marks?

A *mark* distinguishes multiple possible targets.
Kinds of marks

- **Discriminators and enumerators**
  
  ```
  [ isRemote | is Boolean ]
  ```

- **Quantities**
  
  ```
  (if numInstances < Q and 
  frequencyOfAccess < F ?
  LinkedList | 
  HashTable )
  ```

- **Inputs**
  
  ```
  (Append “db_” to all database operation names)
  ```

- **Other marks**
Marking models

A *marking model* is a way to declare:

- Names of marks
- Where they belong in the metamodel
- Their types.

Invocation: Accessibility ::= 
\[
[ \text{isRemote} \ | \ \text{is Boolean} ] = \text{isRemote}
\]

ClassExtent: StorageType ::= 
\[
(\text{if numInstances} < Q \ & \ & \text{frequencyOfAccess} < F \\
\quad ? \text{LinkedList} \\
\quad | \text{HashTable} ) : \text{int}
\]
Table of contents

1. What’s the problem?
2. Models
3. Metamodels
4. Mappings
5. Marks
6. Representing models
7. Agile MDA
8. Conclusion
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
Figure 12-99: A simple EJB profile
Superstructure submission
Table of contents

1. What’s the problem?
2. Models
3. Metamodels
4. Mappings
5. Marks
6. Representing models
7. Agile MDA
8. Conclusion
Elaborative development

Analysis

Intermixed Application and Design

Preliminary Design

Implementation Details and Code Bodies

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

Detailed Design

Target Code

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

A design-time interoperability bus
What’s the solution?

Connect up the models according to:

- a single set of mapping rules that
- operate on to the same metamodel
MDA needs a way to map data from a metamodel into text.

We call them “archetypes”.

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

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

The archetype language produces text.

```plaintext
public:
    enum states_e
    { NO_STATE = 0 ,
    .for each state in stateS
        .if ( not last stateS )
            ${state.Name} ,
        .else
            NUM_STATES = ${state.Name}
        .endif
    .endfor
};
```

```plaintext
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 design-time interoperability bus

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
Examples

**Financial system**
- Highly distributed
- Concurrent
- Transaction-safe with rollback
- Persistence, with rollback
- C++

**Embedded system**
- Single task
- No operating system
- Limited persistence capability
- Optimized data access and storage
- C

**Telecommunication system**
- Highly distributed
- Asynchronous
- Limited persistence capability
- C++

**Simulation system**
- Mostly synchronous
- Few tasks
- Special-purpose language: “Import”
All domains are translated

SAME models on each platform!

Design is specific to category of platforms
Building the system

Generate deliverable production code.

Application Models → Compile the Application Models → Code for the System

Libraries, Legacy or Hand-written code

Model Compiler

Archetypes (Weaving rules)
Run-Time Library (Mechanisms)
Retargeting the environment

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

Realized in thin systems

Realized in General Purpose Computers

Realized in Silicon
Table of contents

1. What’s the problem?
2. Models
3. Metamodels
4. Mappings
5. Marks
6. Representing models
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, which can map between models.
- QVT = Query/View/Transform, a standard for mapping between (MOF) metamodels.
  - This is presently an RFP (request for proposal), and not yet a standard.
- XMI = XML Model Interchange.
## MDA standardization

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Date</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>UML 2.0 Infrastructure</td>
<td>Jan 2003</td>
<td></td>
</tr>
<tr>
<td>QVT (metamodel-metamodel)</td>
<td>Mar 2003</td>
<td></td>
</tr>
<tr>
<td>Marks</td>
<td></td>
<td>Understood</td>
</tr>
<tr>
<td>Action Language</td>
<td></td>
<td>Necessary?</td>
</tr>
<tr>
<td>Archetypes (metamodel-text)</td>
<td></td>
<td>Not yet</td>
</tr>
</tbody>
</table>

The ADTF and the MDA WG proposes these RFPs.
See also

*MDA Distilled*, Mellor, Scott, Uhl and Weise
Addison-Wesley, 2003

*Executable UML*, Mellor and Balcer,
Addison-Wesley, 2003

www.omg.org
www.projtech.com
MDA Distilled

- Started in earnest in March 2002
- First four chapters sent for review in July 2002
- Chapters 5-9 sent for review February 2003
- Meeting to complete last five chapters June 2003
- Review complete by July 2003
- “I have scheduled your book to go into production on 8/1/03.”
  (i.e. 2003-08-01)
Accelerating development of high-quality systems.

Makers of BridgePoint ® Modeling Tools

Stephen J. Mellor
Project Technology, Inc.
http://www.projtech.com