Building Compilable Class Models

David S. Frankel
Chief Consulting Architect
IONA Technologies
david.frankel@iona.com
The Challenge

• **Requirement**: Preserve investment
  – As new platforms emerge
  – As platforms themselves change
    • EJB™ 1.1 → EJB 2.0
    • XML → XML Schema
    • MTS → COM+
    • CORBA™ 2.X → CORBA 3.0

• **Solution**: Isolate information and processing logic from technology specifics
  – Build platform-independent models
  – Map these models to specific platforms
Unified Modeling Language™

- UML™ is independent of
  - CORBA
  - COM
  - EJB
  - XML
  - Etc.
MOF™ Background

- Standard Passed by OMG, 1997
- Standard Constructs for Describing metamodels
- Premise: There will be more than one metamodel
- Sun JSR-40 defining Java Metadata Interface (JMI)
XMI’s UML-XML Mapping Rules

Some slides appear in upcoming Wiley book “Model-Driven Development” by David Frankel and Jack Greenfield
© Copyright IONA and John Wiley and Sons, 2001
Platform Independent Business Information Model

<<BusinessEntity>> Account
  id : String
  balance : Float

<<BusinessEntity>> SavingsAccount
  interestRate : Float

<<BusinessEntity>> CheckingAccount
  minBalance : Float

<<BusinessEntity>> PreferredChecking

<<BusinessEntity>> Customer
  socialSecurityNum : String
  name : String
  address : String

1..n +account

1 +customer
Mapping the Business Information Model to XML

XMI’s UML-XML Mapping Rules

XML DTD (or Schema)

<<BusinessEntity>>
Customer
SocialSecurityNum : String
name : String
Address : String

...
Don’t…

• Specify basic accessor and mutator operations
  – Simply declare attributes and navigable associations
  – Let language mappings produce accessors and mutators
Example

```
<table>
<thead>
<tr>
<th>Class</th>
<th>Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>BusinessTxUnit</td>
<td>+executor: int</td>
</tr>
<tr>
<td></td>
<td>+reverser: int</td>
</tr>
<tr>
<td></td>
<td>+compensator: int</td>
</tr>
<tr>
<td>BusinessTx</td>
<td>+unit: int (ordered)</td>
</tr>
<tr>
<td></td>
<td>+executor: int</td>
</tr>
<tr>
<td></td>
<td>+reverser: int</td>
</tr>
<tr>
<td>CompensatableUnit</td>
<td>+compensator: int</td>
</tr>
<tr>
<td>ReversibleUnit</td>
<td>+reverser: int</td>
</tr>
<tr>
<td>ModelElement</td>
<td>name: String</td>
</tr>
<tr>
<td>Tx</td>
<td>technologyKind: String</td>
</tr>
<tr>
<td></td>
<td>connectionInfo: String</td>
</tr>
</tbody>
</table>

ACIDTx
```
Mapping Attributes

General Mapping Rule

Contract of ModelElement supports getting and setting the value of the name, usually by generating accessor and mutator operations.

JMI Mapping Rule

```java
public interface ModelElement {
    public java.lang.String getName();
    public setName(java.lang.String newValue);
}
```
Mapping
Navigable Association Ends

General Mapping Rule

JMI Mapping Rule

Contract of BusinessTxUnit supports getting and setting the value of the reference to ACIDTx.

public interface BusinessTxUnit extends ModelElement
{
    public ACIDTx getExecutor();
    public setExecutor (ACIDTx newValue);
}
Mapping
Non-Navigable Association Ends

General Mapping Rule

JMI Mapping Rule

Contract of ACIDTx does not support getting and setting the value of the BusinessTxUnit

public interface ACIDTx extends Tx
{

}
Mapping Navigable, Multi-Valued Association Ends

Contract of BusinessTx supports getting and setting the value of the reference to BusinessTxUnit. Mapping differs if not ordered.

General Mapping Rule

JMI Mapping Rule

public interface BusinessTx extends Tx {
    public java.util.List getUnit();
    public void setUnit(java.util.List newValue);
    public void addUnit(BusinessTxUnit newElement);
    public void modifyUnit(BusinessTxUnit oldElement, BusinessTxUnit newElement);
    public void removeUnit(BusinessTxUnit oldElement);
    public void addUnitBefore(BusinessTxUnit newElement, BusinessTxUnit beforeElement);
}
Avoid Name Clashes

**Not Ok.** Using same association end name means two properties of Customer have the same name.

**Ok.** Different association end names distinguish the two properties of Customer.
• MOF (not UML 1.x) allows the modeler to specify whether a client can change the value of an attribute or association end
  – isChangeable tagged value
  – If true, no mutator is generated
Do…

• Consider navigability of association ends carefully
• Use aggregations…properly
• Specify multiplicities on both sides of all associations
• Name all associations and association ends
  – Not all names need to show on diagrams
    • Convention: Suppress display of association names
    • Convention: Suppress display of non-navigable end names
• Use abstract classes where appropriate
Navigable Association Ends Imply Dependencies!

Ok. *Main* remains independent of *Secondary*. Contract of *Y* has `getX` and `setX`.

Not Ok. *Main* is actually dependent on *Secondary*. Contract of *X* has `getY` and `setY`.

Neither end is navigable: Ok
Both ends navigable: Not Ok
Composite (Strong) Aggregation Semantics

- An instance of D can be owned by only 1 instance of C
- Multiplicity of C can be 0..1 or 1..1, so it must be specified!!
  - 0..1 means that a D can exist without being linked to a C
  - But if a D is linked to a C, then it is owned by that C, i.e. its lifetime cannot extend beyond the lifetime of that C
- Multiplicity of D unconstrained
- A.k.a “black diamond”

```plaintext
C
  ↓
D
  ↓
c1 : C
d1 : D
Delete c1
```
No Aggregation Semantics

How does this differ from composite (strong) aggregation?

Execute an operation

Aggregation would not allow this link change
Beware of Shared (Weak) Aggregation

- One possible interpretation
  - An instance of B can be owned by 1 or more instances of A
  - Multiplicity of A can be 0..* or 1..*
  - Multiplicity of B unconstrained
  - Less common than composite (strong) aggregation
  - A.k.a. “white diamond”

Delete a2

Delete a1
public interface BusinessTx_ACIDTx extends javax.jmi.reflect.RefAssociation {
    public boolean exists(ACIDTx executor, BusinessTxUnit businessTxUnit);
    public ACIDTx executor(BusinessTxUnit businessTxUnit);
    public void add(ACIDTx executor, BusinessTxUnit businessTxUnit);
    public void addBeforeExecutor(ACIDTx executor,
                                   BusinessTxUnit businessTxUnit,
                                   ACIDTx before);
    public void addBeforeBusinessTxUnit(ACIDTx executor,
                                          BusinessTxUnit businessTxUnit,
                                          BusinessTxUnit before);
    public void modifyExecutor(ACIDTx executor,
                               BusinessTxUnit businessTxUnit,
                               ACIDTx newExecutor);
    public void modifyBusinessTxUnit(ACIDTx executor,
                                      BusinessTxUnit businessTxUnit,
                                      BusinessTxUnit newBusinessTxUnit);
    public void remove(ACIDTx executor, BusinessTxUnit businessTxUnit);}

Name all associations and ends

ACIDTx +executor 1

BusinessTx_ACIDTx +businessTxUnit 0..n

BusinessTxUnit
Mapping Abstract Classes

**General Mapping Rule**

Separate instance management interface is always generated. But if the class is abstract (denoted by italicized name), it has no create operations.

**JMI Mapping Rule**

```java
public interface TxClass extends javax.jmi.reflect.RefClass {
}
```
Mapping Non-Abstract Classes

JMI Mapping Rule

```java
public interface ACIDTxClass extends javax.jmi.reflect.RefClass {
    public ACIDTx createACIDTx();
    public ACIDTx createACIDTx(java.lang.String name, java.lang.String technologyKind, java.lang.String connectionInfo);
}
```

General Mapping Rule

- Separate instance management interface is always generated. But if the class is abstract (denoted by italicized name), it has no create operations.

JMI Mapping Rule

```java
public interface ACIDTxClass extends javax.jmi.reflect.RefClass {
    public ACIDTx createACIDTx();
    public ACIDTx createACIDTx(java.lang.String name, java.lang.String technologyKind, java.lang.String connectionInfo);
}
```
Using UML to Create MOF Metamodels: Don’t Use…

- Association classes
- N-ary associations (associations among more than two classes)
- Qualifiers
Decomposing Association Classes

Decomposes to

NewsService

Person

Subscription
RenewalDate : Date

NewsService

Person

Subscription
RenewalDate : Date
Decomposing an N-Ary Association

N-Ary Association *Negotiation*, Associating Three Classes

Binary Associations Only
Doing Without Qualifiers

• No set formula
• Work around on a case-by-case basis
UML Profile for MOF

- Rules for using UML to define MOF models
- Submitted to OMG as Part of UML Profile for EDOC