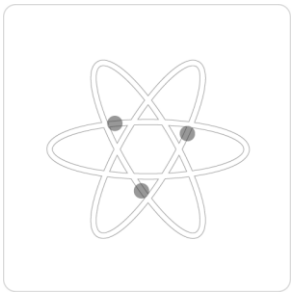
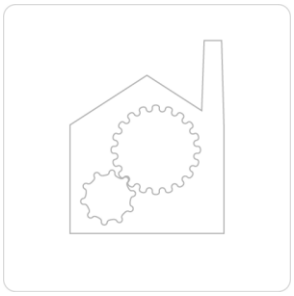
FRAMEWORX™





how to model

Version 0.1

Peer Törngren

2004-09-13

Contents

Disclaimer

THIS PUBLICATION IS PROVIDED ”AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR USE, OR NON-INFRINGEMENT.

THIS PUBLICATION COULD CONTAIN TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERROR. CHANGES ARE PERIODICALLY ADDED TO THE INFORMATION HEREIN; THESE CHANGES WILL BE INCORPORATED IN NEW EDITIONS OF THE PUBLICATION. THE FRAMEWORX COMPANY MAY MAKE NEW IMPROVEMENTS AND/OR CHANGES IN THE PRODUCTS AND/OR THE PROGRAMS DESCRIBED IN THIS PUBLICATION AT ANY TIME.

ALL TRADEMARKS ARE RECOGNIZED.

This is a preliminary product specification.
Specification and Product is subject to change without notice.

Utility names, File names, detailed command lines are likely to change in the final release and are only used in this specification to illustrate the functionality.

Record of Changes and Reviews

Changes

Date	Author	Version	Comments

Reviews

Date	Author	Version	Comments

Intro: Build and Generate

Models

Model Manager operates at UML level. At this level we have 3 models and 1 package of interest.

IM – the Information Model
Mandatory. Holds all IM entities.
AM – the Adapter Model
Optional. Holds model adapters specifying how to adapt IM entities to DM classes (tables).
DM – the Data Model
Optional. Holds DM classes (tables).
Data types
Mandatory. Holds all data types supported by the runtime, both native types and Object types, and also OID strategy classes. The package itself is stereotyped with «modelLibrary» and «topLevel».

Build

When you build, model manager will read the XMI file and generate a complete Adapter Model and Data model for it, using default rules. It will not replace any existing elements. Hence, you can gradually vary the amount of default and custom generation by simply adding and removing specifications in the various models. Whatever is missing will be generated using default rules.

If you want to, you can run model manager to save the generated file (use the model manager 'export' command to do this), and load it into your UML tool (MagicDraw) for inspection and/or editing. You may save your edited result and use as input for future generator runs.

The limitations here have not yet been fully explored, so we may want to be careful with what we promise.

UML Profile

We depend on a model that have all our UML extensions loaded. These are defined in the "InformationModel.xml" (a template model file), and in a separate document. The document describes the meaning of the stereotypes and the syntax of specifying tagged values.

Default Generation

Pre-Condition

Get access to the template model “InformationModel.xml”.

Create Information Model

Copy the template model to some convenient place, using the name you want.

Open it in MagicDraw, find the skeleton Information Model.

Change the name of the Information Model, give it the name you want your schema to have.

Optionally set the tagged value “oidStrategy” to define a non-standard oid strategy.

Create Information Model Entities

Create your IM classes just as before.

For each class, optionally specify

Tagged value {indexHint}
To generate index for specified attributes. Name must be precise, only one per class. See UML profile for how to specify attributes.
Constraint {unique}
to generate constraints in unique columns. NA
Tagged value {extent}
to specify a specific extent name for the class in DB. The default is to use class name and add a plural ‘s’, e.g. Person=>Persons.

Non-standard Generalization Pattern

Pre-Condition

Have an Information Model.

Have at least one generalization (superclass).

Create Adapter Model

Create an Adapter Model at “root” level (UML ‘model’, not package)

Set stereotype «ma» on the adapter model

Create a class diagram in AM root.

Add IM and AM to the diagram.

Connect the AM to the IM with a Realization – MagicDraw has a button for it (Abstraction is a subtype of Dependency, a realization is an Abstraction with «realize»). The arrow must go from AM to IM, the AM “realizes” the IM.

Create Model Adapter classes

In the AM, create a package structure matching the IM package where the interesting element(s) reside, i.e. the superclass and subclass(es).

In the desired package, create a class with the same name as the corresponding IM class.

Set stereotype «ma» on class

Connect each adapter to the corresponding IM class using a regular association. No need for names, role names or multiplicity. To be precise, association should be uni-directional from «ma» to «im», but this has (yet) no semantic impact.

Specify generalization pattern

Width

This pattern is default for an abstract «im» superclass. To apply for a concrete superclass:

Add stereotype «union» to the «ma» superclass
(if superclass has concrete “intermediate” subclasses (subclasses with subclasses), do the same for each intermediate subclass)

Split

This pattern is default for a concrete «im» superclass. To apply for an abstract superclass:

Add stereotype «delegate» to the «ma» superclass.

Add stereotype «join» for each «ma» subclass
(if superclass has abstract “intermediate” subclasses (subclasses with subclasses), do the same for each intermediate subclass)

Depth

This pattern is not applied by default. To apply for a superclass:

If superclass is abstract: add stereotype «delegate» to the «ma» superclass.

Add stereotype «constraint» for each «ma» subclass
(if superclass has “intermediate” subclasses (subclasses with subclasses), do the same for each intermediate subclass)

Non-standard Adapters

DISCLAIMER

None of these features have been thoroughly tested. Before we make any commitments, we must make sure that they work as described … (sorry, this is not a satisfactory condition, I know …)

Pre-Condition

Have Information Model.

Have Adapter Model.

Have at least one model adapter class («ma») connected to an IM entity («im»).

Modify Class/Table name mapping

Rename the model adapter. The generated table will have the name of the adapter, not the im class.

Modify Attribute mapping

Add an attribute with the same name as the IM attribute.

Set the initial value to the name of the DM attribute (the column name).

Set the type to the type of the IM attribute

Optionally specify a custom type adapter in the tagged value {adapter} **(not yet fully implemented, I think).**

Set the multiplicity to the “undefined” or 1 (‘1’ implies ‘NOT NULL’ in db).
(beware of changing to undefined if IM specifies 1 .. ;-)

Modify OID strategy

OID strategy

Set tagged value {oidStrategy} to define a name for an OID strategy to use for this class. Value must match a class with sterotype «oid» in the Data types package.

Primary Key (optional)

By default, the key attribute and operation settings will be taken from the chosen OID strategy.

Define key column(s) by adding attributes:
set «pk» stereotype
initial value is name of the DM attribute (the db column)
type is type of DM attribute
(name of attribute is irrelevant, default is ‘oid’)
Define key operation by adding one operation:
set «pk» stereotype
set name to desired trigger name (default is oid)
set tagged value {columns} to define key columns

“Master Reference” Key (optional)

This is the key used to navigate from subclass to superclass in a “split” pattern. The key is a PK for subclass and FK to superclass.

By default, the key attribute and operation settings will be taken from the chosen OID strategy.

Define key column(s) by adding attributes:
set «pk» and «fk» stereotypes
initial value is name of the DM attribute (the db column)
type is type of DM attribute
(name of attribute is irrelevant, default is ‘isA’)
Define key operation by adding one operation:
set «pk» stereotype
set name to desired trigger name (default is isA)
set tagged value {columns} to define key columns

Non-standard Data Model

DISCLAIMER

None of these features have been tested. Before we make any commitments, we must make sure that they work as described …

Pre-Condition

Have Information Model.

Have Adapter Model.

Have Data Model.

Have at least one data model class («table») connected to an IM entity («im») thru a model adapter «ma».

Modify Table name mapping

Rename the table class. This makes little sense since the same can be achieved by renaming the adapter. It has not been tested.

Add custom triggers and constraints

Add index

Add an operation to the «table» class.

Set stereotype «index»

Set any name, avoid conflict with generator; default name is classname+IX+sequence number, e.g. EmployeeIX, EmployeeIX1, EmployeeIX2, …

Define concerned columns in TaggedValue {columns}

Add unique constraint

Add an operation to the «table» class.

Set stereotype «unique»

Set any name, avoid conflict with generator; default name is classname+UQ+sequence number, e.g. EmployeeUQ, EmployeeUQ1, EmployeeUQ2, …

Define concerned columns in TaggedValue {columns}

Caveats …

Data Model Relationships

Do NOT keep a data model with defined relationships. The generator will NOT detect existing relationships – it will generate duplicates. This is easily fixed, it's just not been done yet.

Generalization patterns and Data Model classes

If you keep a data model class that is involved in a generalization pattern, you must specify some stereotypes on the association(s) between adapter and classes:

- in a "width" pattern, all associations from «union» adapter to dm class must be stereotyped with «union»
- in a "split" pattern, the association from «join» adapter to dm superclass must be stereotyped with «join»
- in a "depth" pattern, the association from «constraint» adapter to dm class must be stereotyped with «constraint»

References

Split Mapping

With split mapping, each class has its own table (storage). The sub-class views combine the sub-class and super-class data.

Figure 2 shows a simplified model of the IM entities, the physical tables, and the mapping-required model adaptors.

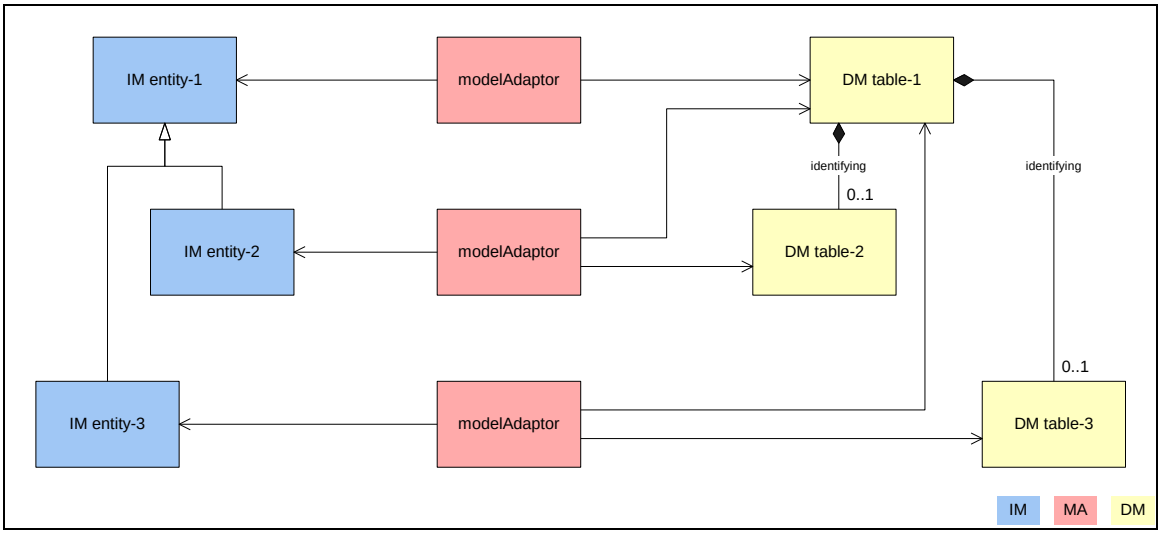


Figure 2 - Spilt Mapping

Width Mapping

Width mapping creates a table for each sub-class, and stores the specific sub-class and super-class data.

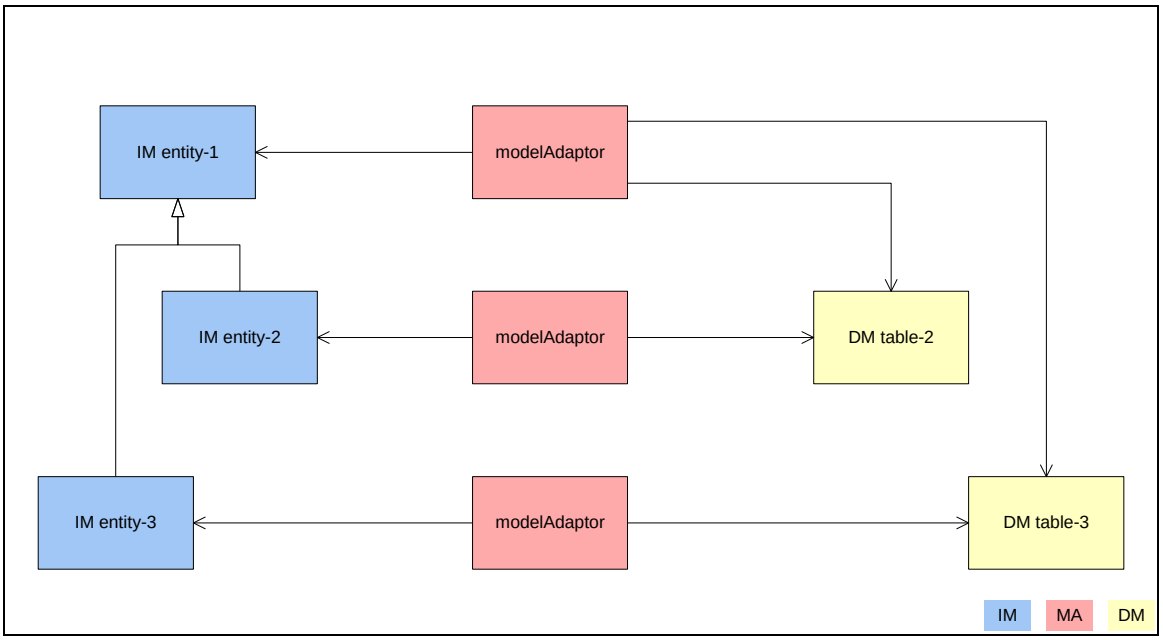


Figure 3 - Width Mapping

Figure 3 shows a simplified model of the IM entities, the physical tables, and the mapping-required model adaptors for width mapping.

Depth Mapping

With Depth Mapping, all sub-classes are stored in a single table that stores both sub-class and super-class data. The sub-class views assist in viewing only the class-specific data, while masking the data of the other sub-classes.

Figure 4 shows a simplified model of the IM entities, the physical tables, and the mapping-required model adaptors for width mapping.

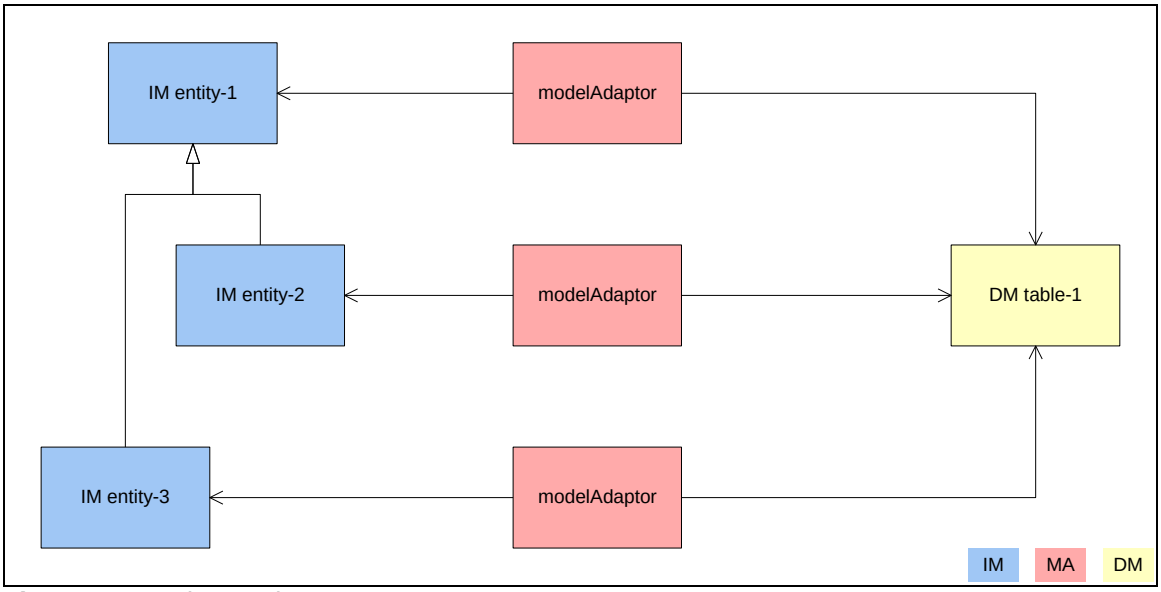


Figure 4 - Depth Mapping

