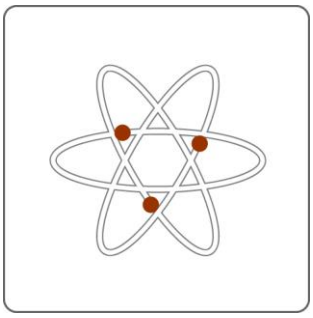
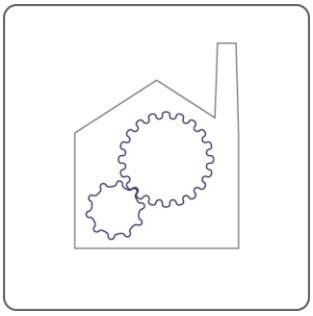
FRAMEWORX™





The Frameworx UML profile for Information and Data Modeling

Version 0.2

Author Peer Törngren

2004-07-28

Content

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.

Record of Changes and Reviews

Changes

Date	Author	Version	Comments
2003-04-06	Peer Törngren	0.1	
2004-07-28	Peer Törngren	0.2	- Cleanup (spelling, wording) - Fix to comply better with skeleton model (`{oidStrategy}, «trigger»`) - Remove refs to build process - Add known issues
2004-09-13	Peer Törngren	0.2	Add {commentTextStyle}, move {id} to new section Generic Extension, ad {derived}

Reviews

Date	Author	Version	Comments

Introduction

About the Document

This document defines the syntax of the **Frameworx UML profile for Information and Data modeling**, a UML Profile for modeling Information Models and Data Models and mapping between these models, as defined by the Frameworx architecture.

The document does not discuss the rationale for the design of the profile, nor how to make best use of the profile.

The document and the profile are intended for public use under the Frameworx license. The profile is designed for use in the context of the Frameworx platform, and this document is written with the same assumption. Some concepts may make little or no sense if used in a generic UML environment.

The document is organized around the models that the profile defines. For each model, all contained elements (packages, classes, associations, attributes, operations, …) are described in terms of extensions (stereotypes, constraints and tagged values) that apply.

Foundation

The Frameworx UML Profile for Information and Data Modeling ("The Frameworx Profile") is based on UML 1.4 and Rational's "UML for Data modeling Profile"[1], and Scott W. Amblers "A UML Profile for Data Modeling"[2].

The Frameworx Profile uses most of the concepts defined by Rational's and Scott W. Ambler's profiles, and adds a few more. The profile also depends on a few standard stereotypes, constraints and tagged values, as defined by the standard UML profile.

The Frameworx Profile is currently provided as a MagicDraw "template model" that is pre-loaded with all stereotypes and packages used in the UML profile. It also includes an empty, skeleton of an Information Model. This model is called `InformationModel.xml`.

License

This document is part of the Frameworx project, and hence subject to the Frameworx Project License. This essentially means that the document may be used, quoted and redistributed freely in its original form, preserving license and copyright statements.

[1] Last known URL was http://www.rational.com/products/whitepapers/437.jsp, now apparently superseded by http://www3.software.ibm.com/ibmdl/pub/software/rational/web/whitepapers/2003/tp162.pdf (by David Gornik).

[2] Last known URL was http://www.agiledata.org/essays/umlDataModelingProfile.html

Frameworx Project License, Version 1.0

Redistribution and use, in source and binary forms, with or without modification, of the software in which this license appears are permitted, provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the disclaimer set forth below.

2. Redistributions thereof in binary form must reproduce such copyright notice, list of conditions and disclaimer in the documentation and/or other materials provided with such redistribution.

3. The end-user documentation included with the redistribution, if any, must include the following acknowledgment:

"This product includes software developed by The Frameworx Company (http://www.frameworx.com/), which may retain copyrights to, and other intellectual property associated with, such software."

Alternately, this acknowledgment may appear in the software itself, if and wherever such third-party acknowledgments normally appear.

4. The name "The Frameworx Company" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact licensing@frameworx.com.

5. THIS SOFTWARE IS PROVIDED "AS IS" AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL ANY DEVELOPER OR DISTRIBUTOR THEREOF OR CONTRIBUTOR THERETO BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Generic Extensions

ModelElement

Tagged Value

- {id}
  the IM entity must be identifiable. Set tagged value {id} to a globally unique identifier (GUID). This is used to track model elements thru transformations:
  id
  a40885d0-67c1-11d7-54a6-0020e063f1ca

- {derived}
  Boolean. A true value indicates that the model element can be completely derived from other model elements and is therefore logically redundant. In an analysis model, the element may be included to define a useful name or concept. In a design model, the usual intent is that the element should exist in the implementation to avoid the need for recomputation. In a Frameworx model, the element is ignored, unless otherwise stated.
  derived
  true

Comment

Tagged Value

- {textStyle}
  An optional, user-defined string to identify the formatting style of comments. This value has no semantic impact on the model, but can be useful for tools that read/write documentation to and from external formats such as XML. Typical values would be "HTML" or "plain".
  commentTextStyle
  html
  Note:
  For technical reasons, this value is currently applied to Model Element rather than Comment.

Datatypes

Package

All data types reside in a single package. It is uniquely stereotyped to be identified without depending on naming conventions. It is located at root level, and is stereotyped with standard UML stereotypes:

- «modelLibrary»
  Defines that this is the data types package
- «topLevel»
  Defines that this is the root of the namespace (representing the “empty” name).

Tagged Value

This package also specifies the default OID strategy. This is defined as the mandatory tagged value {oidStrategy}, where the value is a string referring to an «oid» class in the data types package.

```
oidStrategy
      com.eonworx.oql.mapping.NativeOIDStrategy
```

Class

There is only one stereotype used in the Datatypes package: «oid» marks an OID Strategy that defines how the implicit OIDs in the Information Model are transformed to explicit keys in the physical Data Model.

- «oid»
  Is an OID strategy class. This is a model representation of a class used by the runtime system. It also defines the names of the key attributes and operations.

Information Model

Model

A UML **model** (not a package) stereotyped with «im» denotes a Frameworx Information Model. This model contains all IM entities.

The name of the model defines the IM schema name.

- «topLevel»
  The model is stereotyped as the root of the namespace; all logical names will be created and resolved with respect to the model.

Implicit Defaults

If transforming from «im» to «dm», no defaults are available. All elements must be specified.

If transforming from «dm» to «im», a missing «im» entity is interpreted as an implicit entity with the same name as the «ma» adapter[1], with all features mapped by default rules.

Class

IM entities are stereotyped with the following stereotypes.

- «im»
  The standard im entity
- «imlink»
  A “cross-link” class, existing only to support a many-to-many relationship.
- «powertype»
  Standard UML stereotype.

[1] If the «ma» adapter is also missing, the «im» entity is effectively named after the «dm» table, since this is the implicit name of the missing adapter.

Tagged Value

- `{indexHint}`
  Set tagged value `{indexHint}` to define columns which are good candidates for indices. Specify one or more columns on each line using '+' as a delimiter. Each line suggest an index:
  ```
  indexHint
        firstName+lastName
        phoneNumber
  ```

- `{extent}`
  Defines a non-standard extent for this class. By default, the extent will be the name of the class with a plural *s* added; `Person` will use extent `Person`**`s`**.

Constraint

There is one constraint: `unique`.

This constraint defines one or more attributes that must be unique. Specify as a constraint with a name that starts with 'unique' (e.g. '`uniqueName`', or '`unique12`'), and list all unique attributes on one line, using '+' as a delimiter. Optionally, specify more unique attributes, separating each group by a comma (,).

This is (obviously) not OCL.

```
unique
      firstName+lastName,phoneNumber
```

This defines two unique constraints: the first is *firstName+lastName*, the other is *phoneNumber*.

Association

No stereotypes used, but `«implicit»` associations are ignored.

Attribute

No stereotypes.

Multiplicity

Set multiplicity 1 to define a required (not null) attribute. If undefined, multiplicity 0—1 will be assumed. No other multiplicities are supported.

Tagged Value

Use tagged value `{size}` or `{sizeCode}` to describe size of attribute (depending on type of attribute). `sizeCode` is a symbolic value – see the `java.lang.String` data type for a definition of legal values.

Operation

Not yet used.

Adapter Model

Model

A UML **model** stereotyped with «ma» denotes a Frameworx Adapter Model. This model contains model adapters that transform/adapt IM entities to Data Model classes.

- «topLevel»
  The model is stereotyped as the root of the namespace; all logical names will be created and resolved with respect to the model.

«realize»

The adapter model is connected to an Information Model and a Data Model with two «realize» abstractions (Abstractions with stereotype «realize»). These abstractions define that a particular Adapter Model maps one specific Information Model to one specific Data Model. The direction of the realization is important: the Data model "realizes" the Adapter Model, which "realizes" the Information Model. The arrows flow from Data Model to Adapter Model, and from Adapter Model to Information Model:

[«im» Model] <------ «realize» ----- [«ma» Model] <------ «realize» ----- [«dm» Model]

Tagged Value

The adapter model may specify a tagged value to redefine the default OID Strategy for all adapters in this model. Value must refer to an «oid» class in the Data types package. If absent, the default specified by the Data types package will be used.

```
oidStrategy
      com.eonworx.oql.mapping.NativeOIDStrategy
```

Implicit Defaults

By default, absent adapters are named after the source model elements in a transformation. If transforming from «im» to «dm», a missing «ma» is interpreted as an implicit adapter with the same name as the «im» entity, with all features mapped by default rules. If transforming from «dm» to «im», an implicit «ma» is named after the «dm» table.

Class

The mandatory stereotype to identify a model adapter is «ma».

Additional stereotypes to define generalization pattern:

- «delegate» (implicit)
  Straight mapping to a DM class. This is the default.
- «union»
  Maps a super class in a “width” pattern.
- «join»
  Maps a concrete subclass in a “split” pattern.
- «constraint»
  Maps a concrete subclass in a “depth” pattern.

Tagged Value

(deprecated)

The adapter may have a tagged value to override the default OID Strategy for this specific adapter. Value must refer to an «oid» class in the Data types package. If absent, the default specified by the Adapter Model or the Data types package will be used.

```
oidStrategy
      com.eonworx.oql.mapping.NativeOIDStrategy
```

Association

Associations between «im» and «ma» are not stereotyped.

Associations between «ma» are stereotyped like the model adapter class, if the adapter specifies some kind of generalization pattern:

- (no stereotype)
  One association between one «ma» and one «dm» class.
- «union»
  One or more associations going from superclass’ «ma» to all «dm» classes used by subclasses.
- «join»
  One association from subclass’ «ma» to superclass’ «dm» class. Subclass «ma» also has one plain association to its own (“primary”) «dm» class.
- «constraint»
  One association going from «ma» to the single «dm» shared by superclass and subclass.

Attribute

Attributes define how «im» attributes and associations map to «dm» classes.

- (no stereotype)
  A plain attribute maps an «im» attribute to a «dm» attribute. The name of the attribute defines which «im» attribute it maps, the initial value of the attribute defines the name of the «dm» attribute. If absent, the «im» attribute maps to a «dm» attribute with the same name.

```
lastName=LNAM
```

- «fk»
  An attribute with stereotype «fk» denotes a relation where the «dm» class holds the foreign key. The initial value defines the name of the «fk» constraint (an «fk» operation on the «dm» class).

- «pk»
  An attribute with stereotype «pk» denotes a relation where the «dm» class holds the primary key. The initial value is irrelevant since there can only be one «pk» on any class. The attribute multiplicity is set to 0, and it has no initial value.

«ma» attributes are defined as “frozen classifier” attributes – these settings have no semantic significance, but reflects the nature of the mapping: it is a static design-time mapping.

Tagged Value

Use tagged value {adapter} to specify a non-standard mapping of the attribute. The value may identify a regular data type or a non-standard mapping class (specified by fully qualified name). If specified for a «pk» attribute, the adapter must identify an OID strategy implementation.

```
{adapter = com.foo.BarOIDStrategy}
```

Operation

Not yet used.

Data Model

Model

A UML **model** stereotyped with «dm» denotes a Frameworx Data Model. This model contains data model classes representing database tables or views. By default, absent elements are named after the «ma» elements or (implicitly, thru the default «ma» mapping) the «im» model elements.

- «topLevel»
  The model is stereotyped as the root of the namespace; all logical names will be created and resolved with respect to the model.

Implicit Defaults

If transforming from «dm» to «im», no defaults are available. All elements must be specified.

If transforming from «im» to «dm», a missing «dm» entity is interpreted as an implicit table with the same name as the «ma» adapter[1] with all features mapped by default rules.

Class

Classes in the data model are optionally stereotyped with «table» (implicit default) or «view».

Tagged Value

Use tagged values to define non-standard allocation:

- {extent}
  By default, the extent will be the name of the class with a plural *s* added; Person will use extent Person**s** (if transforming from «im» to «dm» extent is explicitly or implicitly defined by the IM entity).

[1] If the «ma» adapter is also missing, the «dm» table is effectively named after the «im» entity, since this is the implicit name of the missing adapter.

- {tablespace}
  (not supported?)
- {indexspace}
  (not supported?)

Association

Association stereotypes have no semantic significance, but can be used for clarity:

- «non-identifying»
  The implicit default (normally not assigned)
- «identifying»
  One class uses its primary key as a foreign key to another class. This is typical for subclasses in “split” pattern.

Role/navigation names

Role names are used to identify the key used to navigate the association. This effectively “reverses” the way role names are used in a traditional, OO diagram; the role name defines the name of the FK in the class it is attached to.

Attribute

The implicit stereotype is «column». Attributes that are used in keys are stereotyped to indicate this. All attribute stereotypes are optional, and have no semantic significance.

- «column»
  The default and implicit stereotype for all «dm» attributes.
- «pk»
  attribute is used in a primary key. This is typically just one column, by default named ‘oid’ (depends on the OID strategy).
- «fk»
  attribute is used in a foreign key. This is typically just one column (depends on type of primary key), by default named after the navigation name in the Information Model.
- «ufk»
  attribute is used in a “union” foreign key (see more on operation «ufk»)
- «ak»
  attribute is used in an alternative key (not yet used).

Multiplicity

Set multiplicity 1 to define a required (not null) attribute. If undefined, multiplicity 0..1 will be assumed. No other multiplicities are supported.

Tagged Value

Use tagged value `{size}` or `{sizeCode}` to describe size of attribute (depending on type of attribute). `sizeCode` is a symbolic value – see the `java.lang.String` data type for a definition of legal values.

Operation

In the data model, class operations define constraints on the class. The most common constraints are key constraints (primary or foreign), and also unique constraints and indexes.

Key constraints

- `«pk»`
  Primary key. Specify name of attribute(s) in tagged value `{columns}`. This is typically just one column. Using the default native OID strategy, Frameworx will generate an "Object ID" column.
  ```
  columns
        oid
  ```

- `«fk»`
  Foreign key. Specify name of attribute(s) in tagged value `{columns}`. This is typically just one column (depends on type of primary key), by default named after the navigation name in the Information Model.
  ```
  columns
        customer
  ```

- `«ufk»`
  A "union" foreign key. A foreign key cannot refer to a key in a union of multiple tables (e.g. a relation to a superclass adapted using the "width" pattern). This key defines the concerned columns, but the referential integrity constraints are maintained at the other end of the relationship, by each concrete subclass of the superclass. The specification is similar to the `«fk»`:
  ```
  columns
        role
  ```

- `«ak»`
  An alternative key (not yet used).

Delete Constraints

We use stereotypes to declare triggers that ensure referential integrity on delete. Each stereotype is listed with the required tagged values (if any) it may or must use.

Foreign key constraints:

- `«cd»` Cascading Delete
  Place on required `«fk»` (NOT NULL): delete me with foreign element (`«pk»` counterpart has multiplicity 1).

- `«cv»` Cascading Veto
  Place on required `«fk»` (NOT NULL). Reject delete of primary key element if I refer to it (`«pk»` counterpart has multiplicity 1..1 or 1..*).

- «cr» Cascading Release
  Place on optional «fk». Clear reference if «pk» counterpart is deleted («pk» counterpart has multiplicity 0..1 or 0..*).

Referential constraints:

In some cases it is not possible to create a regular foreign key, but we still need to maintain the same kind of constraints. So we add a "referential constraint" that enforces the same kind of behavior as the different foreign key stereotypes, but we place the constraint on the primary key side of the relation. We name the constraint to uniquely identify the concerned counterpart by referring to the Class and the name of the foreign key maintaining the relationship:

```
Class.keyName
```

The key of the concerned class defines the concerned columns.

- «rd» Referential Delete
  Delete referring element with me (same as «cd» for a foreign key)

- «rv» Referential Veto
  Reject delete if I am referred to (same as «cv» for a foreign key).

- «rr» Referential Release
  Clear remote reference if I am deleted (same as «cr» for a foreign key).

Unique constraints

Unique constraints define one or more columns that must be unique. When generated, these are named after the class, with a suffix to mark the type of the constraint, and a sequence number.

Columns are specified as tagged value {columns}, where columns are defined as a discrete value (each value on its own "line" of the tagged value's value definition).

There are several type of "unique" constraints, each identified by a stereotype:

- «unique»
  Basic constraint, ensures that one or more columns are unique. Each value specifies a column that needs to be unique.
  ```
  columns
        firstName
        lastName
  ```

- «unionunique» Union Unique
  columns[1..*]
  Ensure columns are unique across multiple tables. Typical use is in "width" mapping. Each value specifies one table with a unique constraint (applicable to one or more columns). If multiple unique constraints apply, multiple triggers are defined. Concerned tables/classes are listed with Classname.column:
  ```
  columns
        Employee.firstName
        Employee.lastName
        Customer.firstName
        Customer.lastName
  ```

- «typeunique» Type specific Unique
  type[1]
  columns[1..*]

'columns' must be unique if row represents a class of 'type'.
```
type
      Person
columns
      firstName
      lastName
```

Multiplicity Constraints

If a superclass is mapped to a single class that it shares with its subclasses, we cannot use straight multiplicity to enforce NOT NULL constraints. Columns that are required in as subclass are not required in the superclass.

We use a “type specific” constraint to define this:

- `«typerequired»` Type specific required
  `type[1]`
  `columns[1..*]`
  'column' NOT NULL constraint applies only if row represents a class of 'type'.
  ```
  type
        Person
  columns
        firstName
        lastName
  ```

Index

An index is defined by an operation with stereotype `«index»`. When generated, these are named after the class, with a suffix to highlight that the operation is an index, and a sequence number.

- `«index»`
  The index columns are defined by tagged value `{columns}`. Each value specifies one column of the index:
  ```
  columns
        firstName
        lastName
  ```

Triggers

A custom trigger is defined by an operation with stereotype `«trigger»`, with implementation details specified as tagged values:

- `{action=…}` When to run trigger? Any combination of UPDATE, DELETE or INSERT
- `{condition=…}` Optional: a boolean condition that defines if trigger should run.
- `{table=…}` Optional: table name
- `{columns=…}` Optional: columns on which trigger operates (same syntax as keys)
- `{time=…}` run `before | after` insert
- `{body=…}` Trigger implementation.[1]

[1] For technical reasons, the body is defined in the documentation field of the operation rather than the tagged value. Editing area is better, and line breaks and indentations are preserved.

Check

A check is defined by an operation with stereotype «check». This is not (yet) supported.

Appendix: Known Issues

Profile Specification Format

The profile is vaguely defined as a proprietary MagicDraw skeleton model. As UML 2.0 and modeling tools evolve and mature, the profile should be specified as a UML 2.0 profile in a tool-agnostic way.

Stereotype Generalization

Stereotypes should be defined in generalization structures. For instance, «ma» should be the parent of the generalization mapping stereotypes («join», «union» and «constraint»).

Semantic Robustness

The current profile does not distinguish between the models when defining class, attribute and operation extensions. Since all extensions are based on the standard UML meta-classes, it is legal to set tagged values intended for «dm» attributes on «im» attributes, and to «dm» attribute stereotypes (such as «pk») on the attribute of an «im» entity (etc.).

To elevate the semantic robustness, the profile could be based on a slightly extended UML metamodel, where the «im», «ma» and «dm» class, operation and attribute stereotypes are replaced with regular meta-classes (sub-classing Classifier, Operation and Attribute respectively). This would allow the profile to define extensions that are relevant to a specific model only;

Another option might be to split the profile in three; «im», «ma» and «dm» profiles, and apply each profile to the corresponding model only, rather than the entire system model.

Examples

Need to insert sample diagram snippets in the document to illustrate the concepts.

Mapping, not Adapting

In the spirit of MDA, the "Model Adapter" and the "Adapter Model" should be renamed to "Model Mapper" and "Mapping Model", i.e. stereotype «ma» becomes «mm».

Rationale

Need to insert some kind of rationale or discussion on the concepts, especially on generalization mapping patterns.

Verify

Need to verify this document against actual profile as defined by the Frameworx MDR and Model Manager components. Suspect some minor differences exist, for instance around DM triggers, extents and table/index space, and also around multi-line tagged values (tagged values with multiplicity >1).

