miUML Metamodel Class Descriptions
Formalization Subsystem

Leon Starr

Sunday, July 3, 2011

mint.miUMLmeta.td.6

Version 1.1.1

Table of Contents

Introduction

This metamodel captures the semantics of Executable UML as described in the book Executable UML: A Foundation for Model Driven Architectures. This document explains how the Class Model Metamodel supports these rules. Justification and explanation of Executable UML rules themselves are beyond the scope of this metamodel.

These descriptions are part of the Executable UML Class Model Metamodel Class Model. Some of the Classes in this model are well known entities defined in [Steve REF]. The beginning of each such description starts with the phrase "Defined in Executable UML". These descriptions do not attempt to duplicate those definitions. But for convenience, concise statements are included here. For further details, please read these books. All of the other classes were invented for the purposes of this metamodel. Full descriptions are included.

Key to Styles and Conventions

COLOR is used to convey information in this document. If you have the ability to print or display this document in color, it is highly recommended. Here are a few notes on the font, color and other styles used in class model descriptions.

Naming classes, types and subsystems

Modeled elements such as classes, types and subsystems are written with initial caps. The text Air Traffic Controller, for example, would probably be a class. If you see the text Application Domain, you know it is a modeled element, most likely a domain.

Naming attributes

Attributes are named with an initial cap followed by lower case, Time_logged_in, for example.

Instance values

Instance data, such as attribute values, are either in quote marks as in "Gwen" and "Owen" or represented in the following color/font: Gwen and Owen.

Whitespace

When programming, the squashedNamingStyle is optimal for easy typing and is adequately readable for short names. That's fine for programming, but analysis is all about easy readability and descriptive, sometimes verbose names. Programming style is retained for method names, but everywhere else whitespace or underscores are employed. For example: Air Traffic Controller, Time logged in or On_Duty_Controller.Time_logged_in.

hardcoreJavaProgrammersMayDisputeThisPointAndArgueThatWhiteSpacesAreEntirelyUnnecessaryForImprovingReadabilityButIdisagree.

Attribute heading color

Attribute headings are colored according to the role the attribute plays in its class. Referential attributes are brown, **Logged in controller**, for example. Naming attributes are blue, **Number**, for example. Finally, descriptive attributes are red, **Time logged in**, for example. Derived attributes are purple **\Volume**, for example.(One of the nice things about color is that it helps eliminate the need for underscores).

Referential renaming

A referential attribute will often have a name that reflects its role rather than the name of the base reference. Station.Logged_in_controller may refer to On_Duty_Controller.ID, for example. Such renaming will be defined in the referential attribute's description.

MDA (Model Driven Architecture) Layers

The MDA defined layers are referenced from time to time. It is sometimes useful, especially in a metamodel, to distinguish the various meta layer names: M0, M1, M2 and M3. These are M0: data values ('N3295Q', 27L, 490.7 Liters, ...) M1: domain specific model elements (class 'Aircraft', attribute 'Altitude', relationships 'R2 - is piloted by'), M2: metamodel elements to define miUML (class: Class, class: Attribute, class: Relationship, attribute:'Rnum'...) and M3: meta-metamodel elements to define (ambitiously) any modeling language (class 'Model Node', class 'Structural Connection'). For our purposes, we will seldom refer to the M3 layer and M2 will be relevant only if the subject matter at hand is a metamodel.

Keep in mind that these layers may be interpreted relative to the subject matter at hand. If you are modeling an air traffic control system, 490.7 Liters is likely data in the M0 layer with the class 'Aircraft' at the M1 layer. 'Class' would be at M2 - the metamodel layer.

But if the subject matter is Executable UML, as is the case in the miUML metamodels, both the class 'Aircraft' and 490.7 Liters could be compressed into the M0 layer. ('Aircraft' is data populating the Class and Attribute subsystem and 490.7 Liters populated into the Population subsystem). At the M1 layer we would possibly have 'Class' and 'Value'. And, since the metamodel will populate itself, as we bootstrap into code generation, the M2 layer is also 'Class' and 'Value'.

In other words, we can create a data base schema from the miUML metamodel and then insert the metamodel itself into that schema. So if we define a table 'Class' based on our metamodel in the data base, we could then insert metamodel classes 'Class', 'Attribute', etc. into that table. Thus we see 'Class' both at the M0 (inserted data) and M1 (modeled) and M2 (metamodel) layers!

Local Data Types

The following data types are used by attributes in this subsystem.

Name

This is a string of text that can be long enough to be readable and descriptive. A handful of words is okay, a full tweet is probably too much. Since the exact value may change to suit a particular platform, it is represented symbolically as "LONG_TEXT".

Domain: String [LONG_TEXT]

Nominal

This is a whole number used purely for naming with no ordinal or computational properties.

Identifiers used only for referential constraints may be stripped out during implementation (assuming the constraints are still enforced). During simulation and debugging, however, these can be nice to have around for easy interpretation of runtime data sets without the need for elaborate debug environments. "File-6:Drawer-2:Cabinet-"Accounting" is easier for a human to read than a bunch of pointer addresses.

Domain: The set of positive integers {1, 2, 3, …}.

Description

This is an arbitrarily long amount of text. The only length limit is determined by the platform.

References

[MB] Executable UML: A Foundation for Model-Driven Architecture, Stephen J. Mellor, Marc J. Balcer, Addison-Wesley, 2002, ISBN 0-201-74804-5

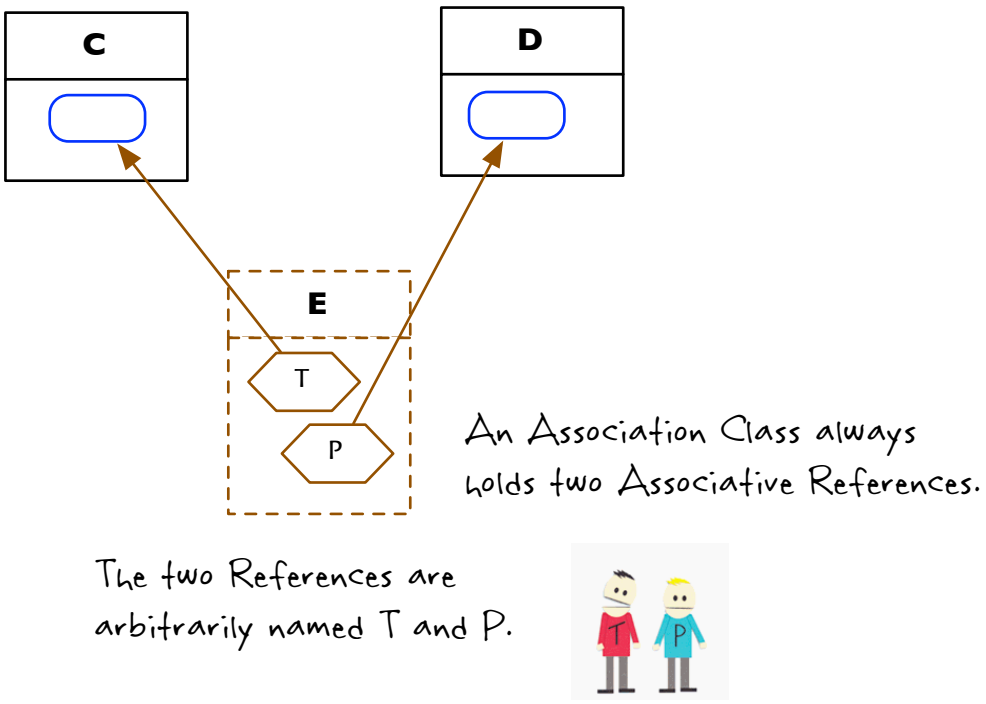
[DATE1] An Introduction to Database Systems, 8th Edition, C.J. Date, Addison-Wesley, 2004, ISBN 0-321-19784-4

[LSART] How to Build Articulate UML Class Models, Leon Starr, Model Integration, LLC, Google Knol, http://knol.google.com/k/how-to-build-articulate-uml-class-models

[OOA96] Shlaer-Mellor Method: The OOA96 Report v1.0, Sally Shlaer, Neil Lang, Project Technology, Inc., 1996 (Available as a PDF download at www.modelint.com)

Associative Reference AREF

An Association Class formalizes an Association by referring to each Perspective, regardless of multiplicity. An *Associative Reference* is a set of Referential Attributes in an Association Class referring to the Identifier on one Perspective in an Association.



Attributes

Type - constrained

Every Associative Reference is named either T or P. These are entirely arbitrary names based on the animated South Park characters 'Terrance and Phillip' who are adequately binary.

Constrained to be either T or P.

Association class

Reference.From class

Participating class

Reference.To class

Rnum

Reference.Rnum

Domain

Reference.Domain

Identifiers

1> Type + Rnum + Domain

Same as the superclass with associative and participating class omitted. Since there can only be one T and one P Reference on an Association Class, the Type is unique within an associative Association.

Relationships

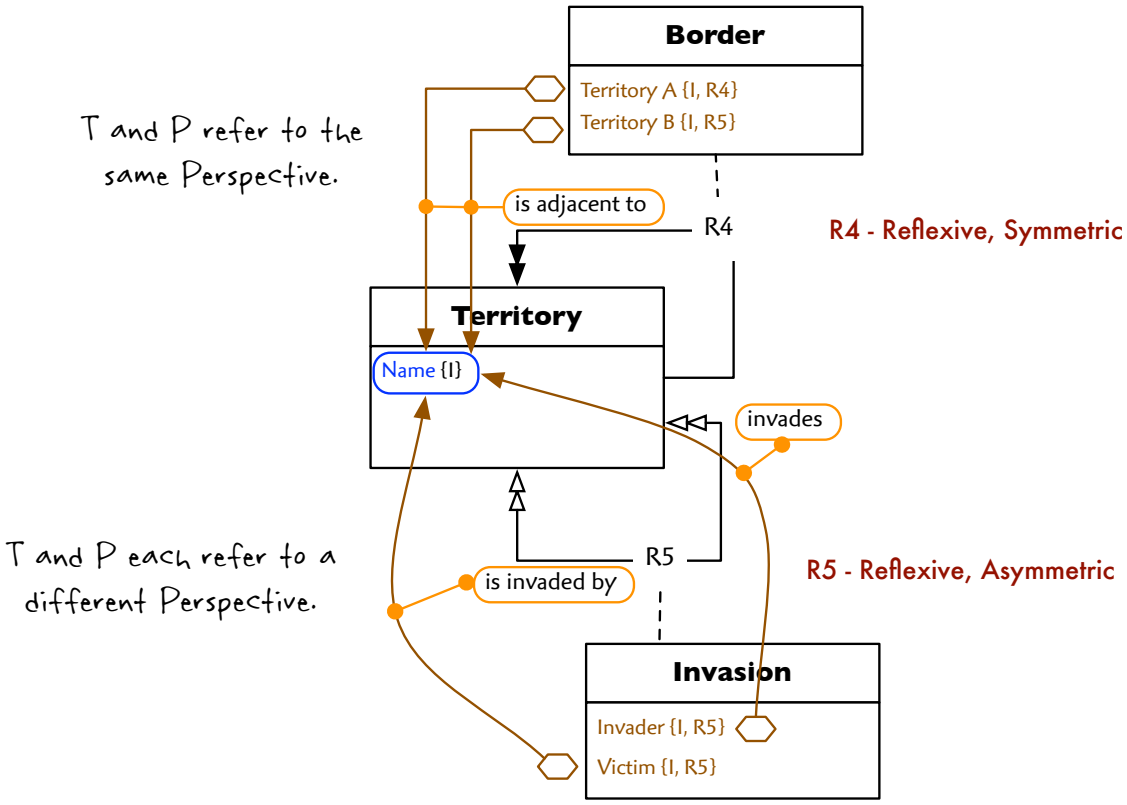
R153
Associative Reference IS A T OR P Reference

Every Association Class must hold two Associative References. Each Associative Reference will refer to a Perspective (the same one in the case of a symmetric reflexive Association). To tell one from the other, each is distinguished arbitrarily as T or P. The T/P specialization makes it possible to express the rule that each Association Class requires exactly one of each.

R175
Associative Reference IS A Symmetric OR AN Asymmetric Associative Reference

As shown below, an Asymmetric Perspective is referred to once, via T or P Reference, in an Association whereas both Symmetric Associative References point to the same Perspective. This constraint is captured by specializing the relationships to each type of Perspective. The Symmetric/Asymmetric Associative Reference specialization makes it possible to state each rule independently.

Differing multiplicity for Symmetric vs. Asymmetric Reference Types



Asymmetric Associative Reference ASYM_AR

This is an Associative Reference that refers to an Active or Passive Perspective on a Binary Association which may or may not be reflexive.

Attributes

Type - constrained

Same as the superclass. Constrained to be the values [T | P]

Association class

Associative Reference.Association class

Participating class

Associative Reference.Participating class

Side - constrained

Asymmetric_Perspective.Side – The Perspective of the Association being referenced. Constrained to being [active | passive]

Rnum

Associative Reference.Rnum and Asymmetric Perspective.Rnum — Both the Reference and the Perspective must be on the same Relationship.

Domain

Associative Reference.Domain and Asymmetric Perspective.Domain — Both in the same Domain.

Identifiers

1> Type + Rnum + Domain

Same as the superclass.

2> Side + Rnum + Domain

There can be only one Reference to the same Perspective in an Association.

RELATIONSHIPS

R174

Asymmetric Associative Reference REFERS TO *exactly one* Asymmetric Perspective
Asymmetric Perspective IS REFERRED TO BY *exactly one* Asymmetric Associative Reference

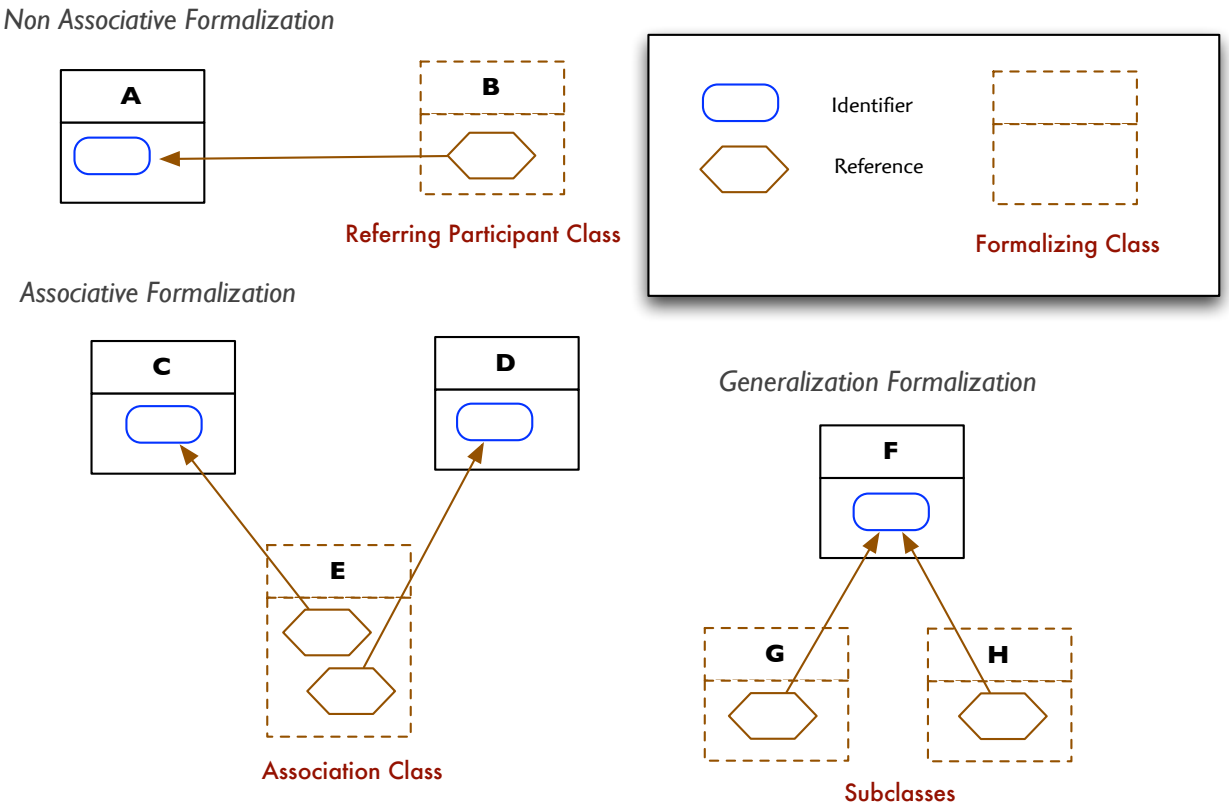
The sole purpose of an Asymmetric Associative Reference is to refer to either the Active or Passive Asymmetric Perspective in the same Association.

An Asymmetric Association is not properly formalized unless it has both T and P references pointing each to either the Active or Passive Perspective in the Association. If T points to the active side, P points to the passive side and vice versa.

Formalizing Class F_CLASS

In miUML, each Relationship is formalized by one or more References to an Identifier from some Class. The number and placement of such References is determined by the type and multiplicity of the Relationship. At its heart, though, the mechanism is always the same. An Identifier in one Class is referred to by one or more Referential Attributes in one or more other Classes. Each Class referring to an Identifier plays a role as a *formalizing class* for the Relationship.

Here is a visual summary of each formalization method:



Attributes

Rnum

Relationship.Rnum — The Relationship to be formalized.

Class

Class.Name — The name of the Formalizing Class. The Referential Attributes will originate here so that any Identifier is referenced from this Class.

Domain

Class.Domain and Relationship.Domain — It is not possible to refer to a Class in a different Domain. So both the Class and Relationship must be in the same Domain.

Identifiers

1> Class + Rnum + Domain

Standard Mx:Mx identifier composition with the Domain referential attributes are merged.

Relationships

R150

Class IS USED TO FORMALIZE *zero, one or many* Relationship
Relationship IS FORMALIZED IN *one or many* Class

In miUML there is only one mechanism to formalize a Relationship and that is for one Class to refer to the Identifier of another. Consequently, there must always be at least one Class holding the necessary Referential Attributes. In the case of a Generalization, multiple Subclasses will refer to the same Superclass. In fact, it is only in the case of a Generalization where more than one Class is necessary to formalize a Relationship.

A Class may simultaneously formalize any number of Relationships. And it is also possible, though rare, for a Class to not formalize any Relationships at all. A Class may, for example, be the referenced Class only or simply not be related to any other Class.

R151

Formalizing Class IS A Subclass OR Participating Class OR Association Class

Each type of Relationship uses a different Formalization Class or configuration of Formalization Classes. A Generalization requires multiple Subclasses all referring to a common Superclass. In this case, each Subclass serves as the Formalizing Class. A non-associative Association uses one of its participating Classes to hold the Referential Attributes. Finally, an Associative Association designates a single Association Class for formalization.

By breaking out each type of Formalization Class, the rules of how References are constructed and managed in each case can be enforced.

P Reference PREF

(See the class description for T Reference).

Attributes

Association class - constrained

Source of the Reference Also constrained to match the Association Class.Class where the Reference is anchored.

Participating class

Associative Reference.Participating class

Rnum

Associative Reference.Rnum and Association Class.Rnum — The Reference and the Association Class must be part of the same Relationship.

Domain

Associative Reference.Domain and Association Class.Domain — All Relationship components are in the same Domain.

Identifiers

1> Rnum + Domain

Same as the superclass minus the Type since it always has the same value, P.

Relationships

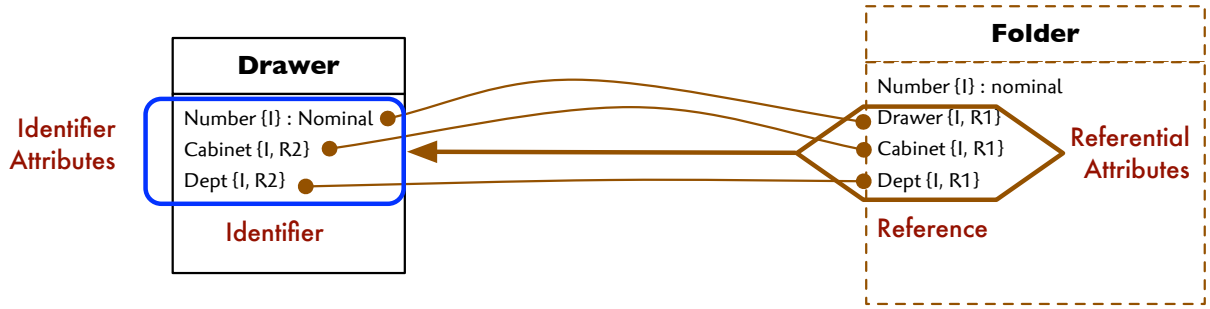
R159
Association Class ANCHORS *exactly one* P Reference
P Reference IS ANCHORED IN *exactly one* Association Class

The Referential Attributes of a P Reference are housed in the Association Class that formalizes the Association.

The Association Class is the location of both T and P References.

Reference REF

An Identifier on one Perspective may be referenced from a Formalizing Class using a set of Referential Attributes in that Class, each of which corresponds to some Attribute of the referenced Identifier. In a Reference, each Attribute of the referenced Identifier must have a corresponding Referential Attribute and each Referential Attribute in the Reference must refer to a distinct Attribute in the Identifier. So there is a complete 1:1 unconditional mapping from Reference Attributes to Identifier Attributes within a Reference.



Attributes

Type

The name of the Subclass of this Reference.

Type: Ref Type [S | O | T | P]

From class

Reference Path.From class

To class

Reference Path.To class

Rnum

Reference Path.Rnum

Domain

Reference Path.Domain

Identifiers

1> Type + From class + To class + Rnum + Domain

No two References of the same Type may follow the same Reference Path. For example, there can be a T and P Reference between the same pair of from-to Classes, but not never two T's or two P's.

Relationships

R160

Reference FOLLOWS *exactly one* Reference Path
Reference Path IS FOLLOWED BY *one or many* Reference

The entire purpose of a Reference Path is to apply the same From-To reference direction to a pair of T/P Associative References. So it stands to reason that a Reference is constrained to follow a single Reference Path.

The referential purpose and definition of an Association Class requires exactly two Associative References along any Associative Path. The 'two-ness' constraint is nailed down by the T and P subclasses, so we use the adequately encompassing 1:M multiplicity here.

R152

Reference IS A Superclass Reference, To One Reference OR AN Associative Reference

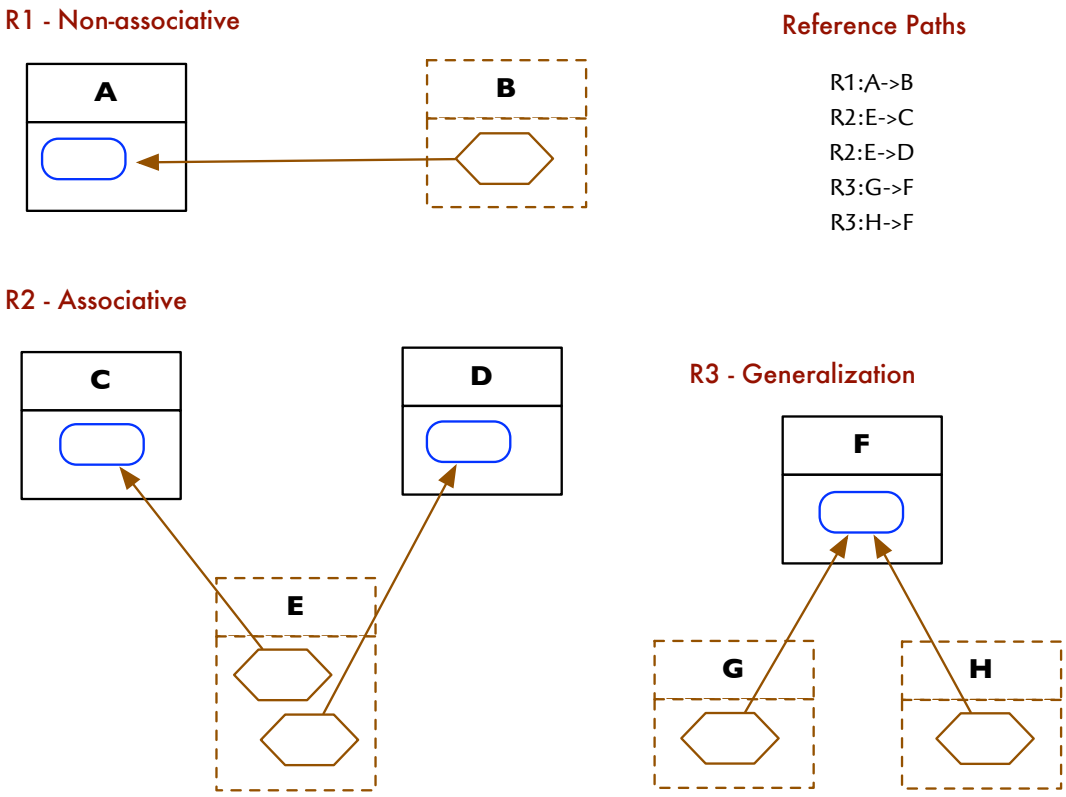
A Reference formalizes a specific type of Relationship: Generalization – Superclass Reference, Non-Associative – To One Reference and Associative – T/P Associative References. This specialization makes it possible to define the relationship between References and Formalizing Classes for each kind of Relationship.

Reference Path RPATH

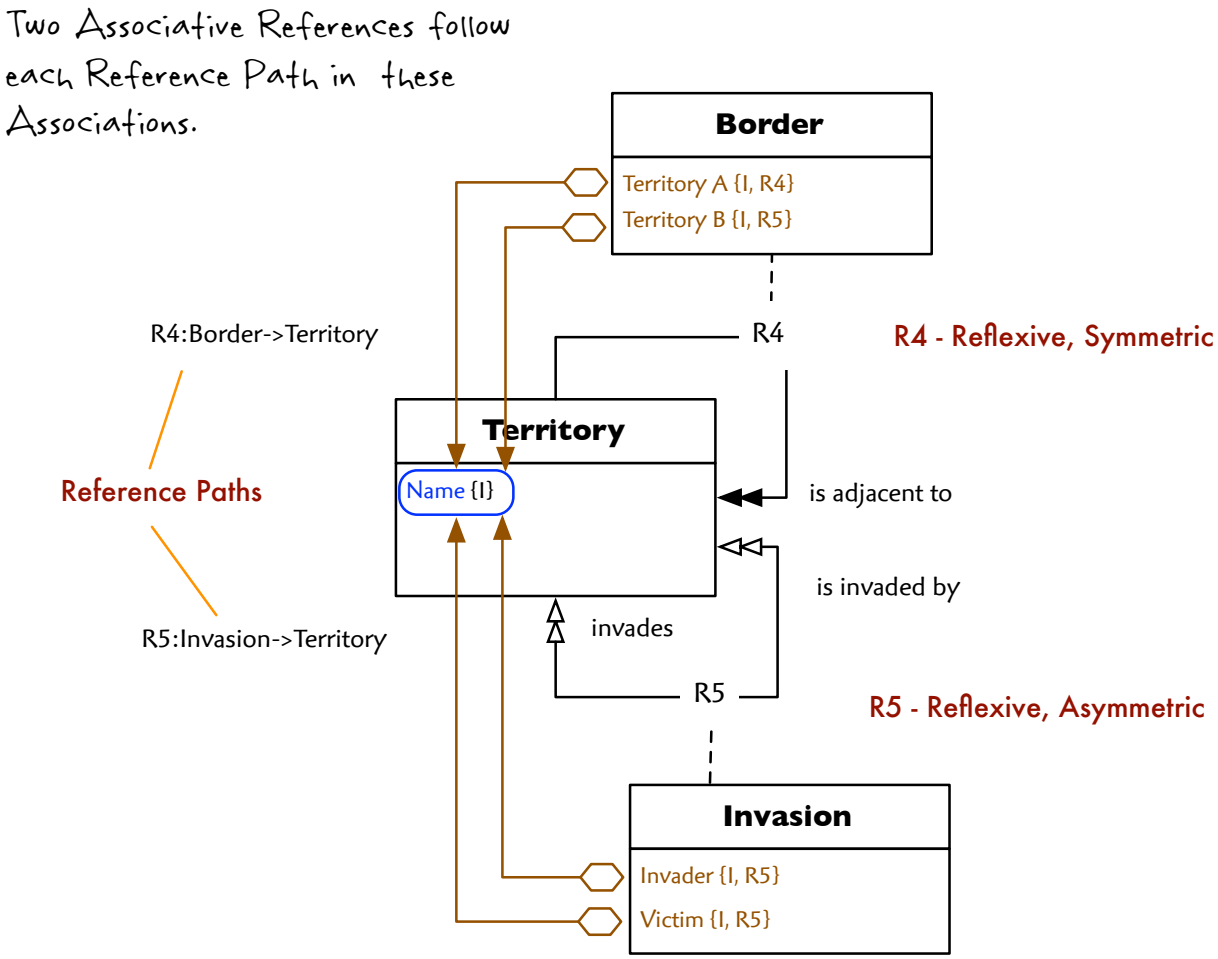
The direction from a referencing Class to the Class it references establishes a *reference path*.

At first glance it seems that a Formalizing Class refers to a Class only once for a Relationship. This is indeed the case in a Generalization, in a non-associative Association and in any non-reflexive Association that uses an Association Class as shown below:

In each of these cases, there is only one Reference per Reference Path.



When a reflexive Association is formalized by an Association Class, however, it becomes necessary to refer to the same Class twice along the same path. (The from-to Classes are the same for each required Reference). The distinction between the two References is established by an arbitrary T/P naming scheme where the T and P References each map to a distinct Perspective on an asymmetric, reflexive Association or both to the same Perspective on a symmetric, reflexive Association as shown below:



So a *Reference Path* is simply a direction from a Formalizing Class to the same or some other Class along followed by one or more References in the same Relationship.

Attributes

From class

Formalizing Class.Name — The reference source.

To class

Class.Name — The Formalizing Class refers to the Identifier in this Class.

Rnum

Formalizing Class.Rnum — The Relationship fully or partially formalized by this Reference.

Domain

Class.Domain and Formalizing Class.Domain — A Relationship may not span Domains.

Identifiers

1> From class + To class + Rnum + Domain

Standard Mx:Mx identifier composition rules with the Domain referential attributes merged.

Relationships

R155

Formalizing Class REFERS TO *one or many* Class
Class IS REFERRED TO FROM *zero, one or many* Formalizing Class

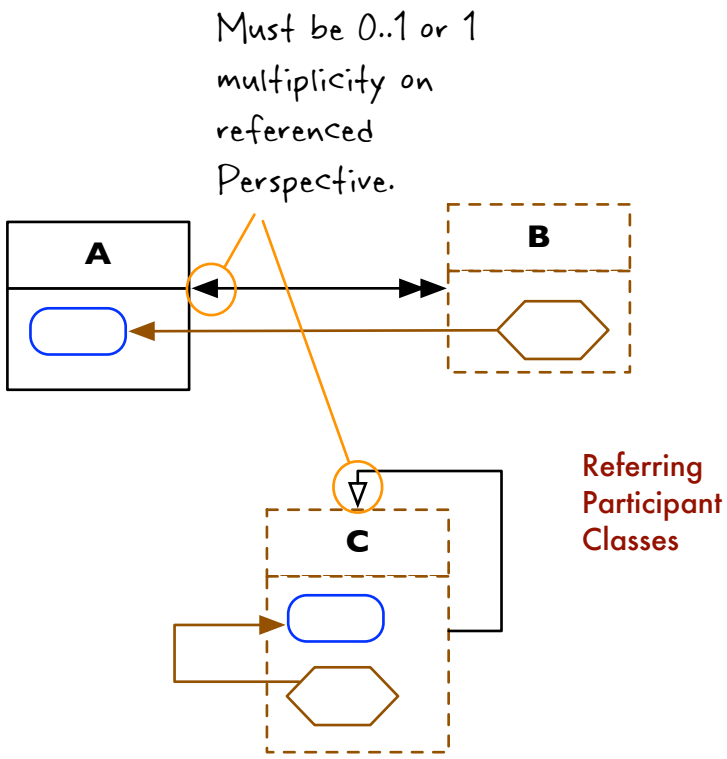
By definition, a Formalizing Class refers to at least one Class. Two Classes are referenced only when an Association Class is the formalization Class on a non-reflexive Association. Two is, in fact, the maximum and this constraint (one or two) is enforced as appropriate in each subclass.

It may be that a Class is not referenced at all. This could be the case if a Class refers to other Classes only or if a Class simply does not participate in any Relationship.

Referring Participant Class RP_CLASS

If an Association does not have an Association Class (non-associative) it must be formalized by one of its participating Classes. The *referring participant class* anchors a Reference to the Identifier in the Class at the opposite Perspective, which must have a 1x multiplicity.

In the case of a non-associative reflexive Association, both Perspectives land on the same Class which must, therefore, be the Referring Participant Class. It refers to itself.



There must be a multiplicity of 1x on the opposing Perspective. This ensures that each Referential Attribute is single valued as required by the relational model normalization rules. This means that the Association multiplicity may never be Mx:Mx. An Mx:Mx multiplicity pair requires the use of an Association Class to formalize the Association.

Attributes

Rnum

Formalizing Class.Rnum

Class

Formalizing Class.Rnum

Domain

Formalizing Class.Domain

Identifiers

1> Rnum + Class + Domain

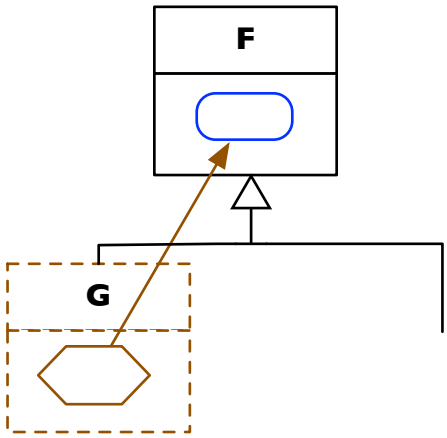
Same as the superclass.

Relationships

None.

Superclass Reference SREF

Each branch of a Generalization Relationship is formalized by a Reference from a Subclass to its Superclass. The Reference is a set of Referential Attributes in the Subclass each of which refers to a component of the Superclass Identifier in a 1:1 mapping. This Superclass Reference must necessarily be an Identifier of the Subclass.



Superclass Reference

Attributes

Subclass

Reference.From class and Subclass.Class — The referring Subclass.

Superclass

Reference.To class and Superclass.Class — The referred-to Superclass.

Rnum

Reference.Rnum, Superclass.Rnum and Subclass.Rnum — The Reference is in the Subclass referring to a Superclass all in the same Generalization Relationship.

Domain

Reference.Domain, Superclass.Domain and Subclass.Domain — A Generalization may not span Domains.

Identifiers

1> Subclass + Superclass + Rnum + Domain

Same as the superclass (Reference Path).

Relationships

R170
Superclass Reference REFERS TO *exactly one* Superclass
Superclass IS REFERRED TO BY *one or many* Superclass Reference

By definition, the Referential Attributes in a Superclass Reference are all pointing toward the corresponding Identifier Attributes of the Superclass in the same Generalization.

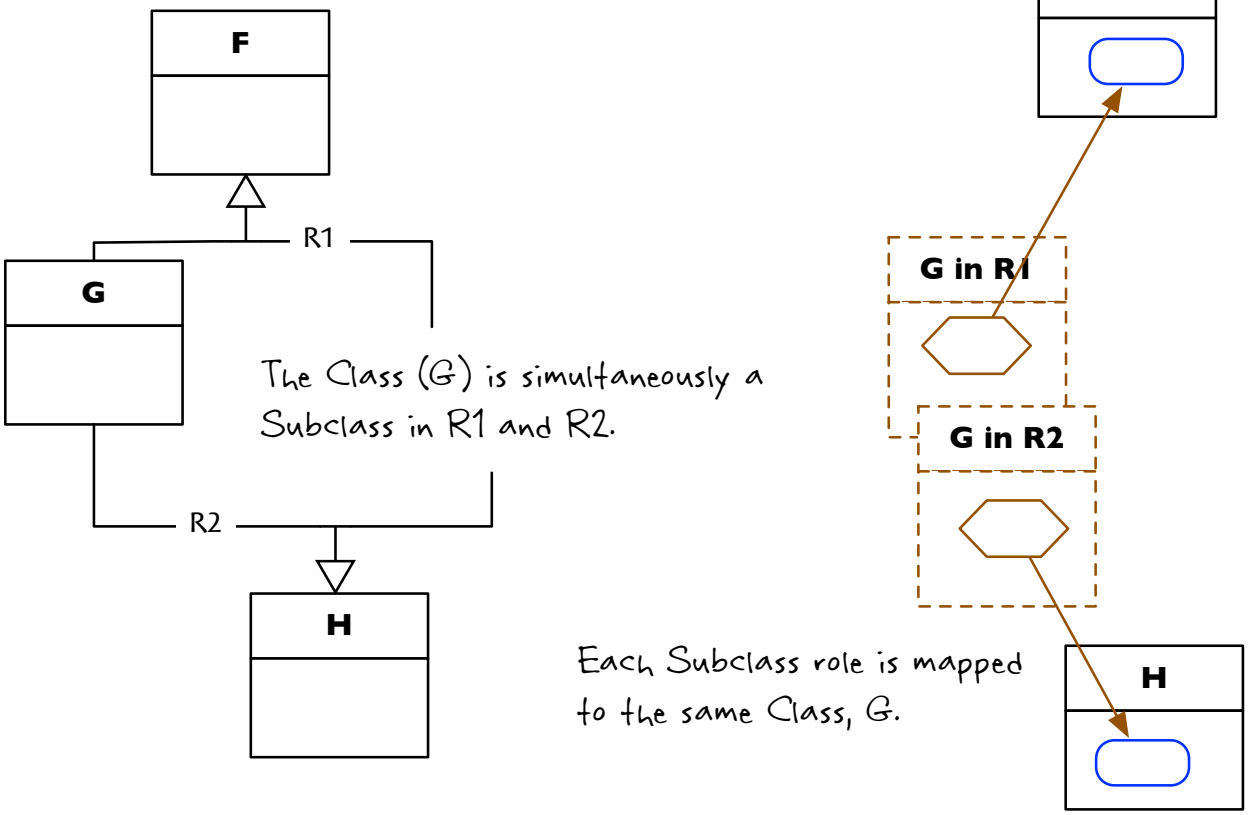
As a consequence of the Minimal Partition constraint, a Superclass must be referred to by at least two Superclass References. In other words, each Generalization has at least two Subclasses, each with a Reference to the same Superclass.

R156
Subclass ANCHORS *exactly one* Superclass Reference
Superclass Reference IS ANCHORED IN *exactly one* Subclass

By definition, a Superclass Reference originates from a single Subclass. In other words, all of the Referential Attributes in the Superclass Reference essentially define the Subclass role.

A Subclass is a formalization role played by a Class. If the same Class participates in multiple Generalizations, there will be a separate role for each Generalization, all tied to the same Class. Each role represents either a Subclass or Superclass.

A Subclass then, is the source of only one Superclass Reference. On the other hand, a Class playing multiple Subclass roles simultaneously would be the source of multiple Superclass References as shown below:


F
R1
G
The Class (G) is simultaneously a Subclass in R1 and R2.
R2
H
F
G in R1
G in R2
H
Each Subclass role is mapped to the same Class, G.

Symmetric Associative Reference SYM_AR

This is an Associative Reference that refers to a Symmetric Perspective on a Unary, reflexive Association.

Attributes

Type - constrained

Associative Reference.Type — Must be [T | P]

Association class

Associative Reference.Association class

Participating class

Association Reference.Participating class

Rnum

Associative Reference.Rnum and Symmetric Perspective.Rnum — This Reference and the Perspective to which it refers is part of the same Association.

Domain

Association.Reference.Domain and Symmetric Perspective.Domain — A Unary Association is entirely within a single Domain.

Identifiers

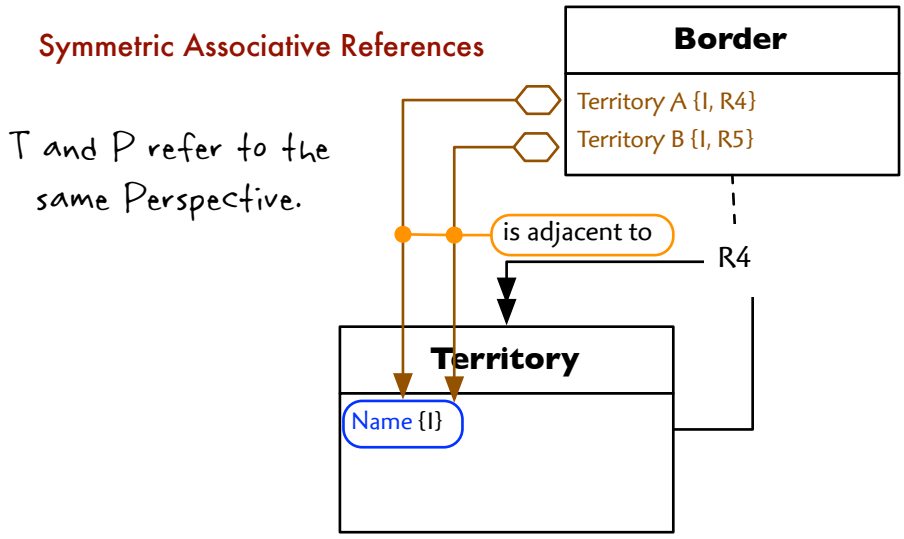
1> Type + Rnum + Domain

Same as the superclass.

RELATIONSHIPS

R173

Symmetric Associative Reference REFERS TO *exactly one* Symmetric Perspective
Symmetric Perspective IS REFERRED TO BY *one or many* Symmetric Associative Reference



The sole purpose of a Symmetric Associative Reference is to refer to the only Symmetric Perspective in the same Association.

A Symmetric Association is not properly formalized unless it has two Symmetric Associative References, both T and P, referring to it.

From the Border / Territory model example we have (T) Territory A and (P) Territory B each referring from the Border Association Class to the Territory Identifier.

T Reference TREF

An Association Class is characterized by two sets of Referential Attributes. Each set refers to a Perspective on the formalized Association. These two sets are arbitrarily named *T* and *P*.

If the Association is asymmetric, T and P will each refer to either the Active or Passive Perspective. If T is Active then P will be Passive or vice versa. On a symmetric Association both T and P will refer to the same (and only) Symmetric Perspective.

Attributes

Association class - constrained

Associative Reference.Association class — Source of the Reference Also constrained to match the Association Class.Class where the Reference is anchored.

Participating class

Associative Reference.Participating class — The T Reference refers to an Identifier in this class.

Rnum

Associative Reference.Rnum and Association Class.Rnum — The T Reference and Association Class are all part of the same Association.

Domain

Associative Reference.Domain and Association Class.Domain — An Association is entirely within a single Domain.

Type - constrained

Associative Reference.Type — Required reference to the superclass. Constrained to the single value [T], by definition.

Identifiers

I> Rnum + Domain

Same as the superclass minus the Type since it always has the same value, T.

RELATIONSHIPS

R158

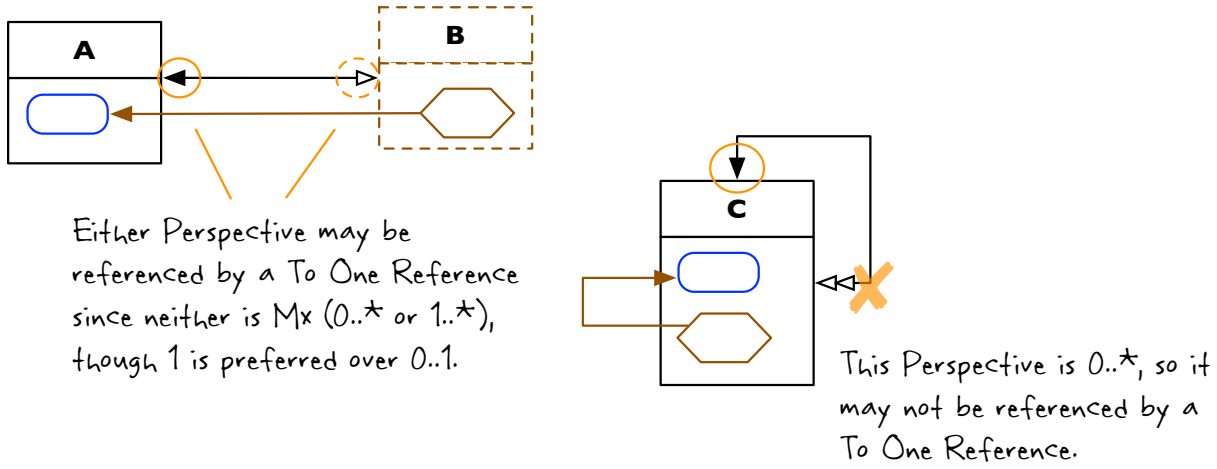
Association Class ANCHORS *exactly one* T Reference
T Reference IS ANCHORED IN *exactly one* Association Class

An Association Class formalizes an Association by pairing two Instances. It is not enough to pair the Instances. It must be clear to which Perspective each Instance belongs. To accomplish this, a T and P Reference is established in each Association Class regardless of any multiplicities on the formalized Association. The T Reference refers to an Instance on one Perspective while the P Reference refers to an Instance on the other. In the case of a Unary Association, T and P will each refer to the same Perspective.

To One Reference T1_REF

When no Association Class is present on an Association (non-associative), a participating Class will be the Referring Class. The Referring Participant Class will house a *to one reference*, so called because the referenced Perspective must have a multiplicity of 1x.

Legal To One References



A Referring Class cannot point to an Mx multiplicity since that would yield a non-normalized multi-valued Referential Attribute. These are not allowed. If an Association is Mx:Mx and, hence, does not have a One Perspective, it must be formalized by an Association Class.

If an Association is 1x:1x, any participant may serve as the Referring Class.

Both reflexive and non-reflexive Associations follow the same rules as shown above.

Attributes

From class

Reference.From class and Referring Participant Class.Class

To class - constrained

Reference.To class — This is the referenced Class which is constrained such that it matches the value found via the path One Perspective[R171]->Perspective[R104].Viewed class. This ensures that the To class is not on an Mx Perspective of an Association.

To side

One Perspective.Side — The referenced One Perspective.

Rnum

Reference.Rnum and One Perspective.Rnum and Referring Participant Class.Rnum — The Reference, referred to One Perspective and anchoring Class are all part of the same Association.

Domain

Reference.Domain and One Perspective.Domain and Referring Participant Class.Domain — An Association is entirely within a single Domain.

Identifiers

1> From class + Rnum + Domain

Same as the superclass (Reference) with Type excluded since it will always be [0]. Also, the To Class is omitted since there is only one Formalizing Class in a non-associative Association.. Also, a reference to a one unconditional side (R157) constitutes an identifier.

2> To class + Rnum + Domain

Also same as the superclass (Reference) with Type and From class excluded since a Class may be referenced only once within a non-associative Association.

3> To side + Rnum + Domain

A reference to a one unconditional side (R171) constitutes an identifier. The To side can take the place of To class as a partial identifier since only one Perspective may be referenced.

Relationships

R171

To One Reference REFERS TO *exactly one* One Perspective
One Perspective IS REFERRED TO BY *zero or one* To One Reference

By definition, a To One Reference refers to the Identifier of a Class viewed from a One Perspective in the same non-associative Association.

A One Perspective must be referenced if it is the only One Perspective in a non-associative Association, i.e. if the Association multiplicity is 1x:Mx. If, on the other hand, a One Perspective is in a non-associative Association where each side is a One Perspective (1x:1x), then only one of these One Perspectives will be referenced leaving the other un-referenced. And if a One Perspective is in an Association formalized by an Association Class, it will be referenced by an Associative Reference and not a To One Reference.

R157

Referring Participant Class ANCHORS *exactly one* To One Reference
To One Reference IS ANCHORED IN *exactly one* Referring Participant Class

A To One Reference, by definition, is a Reference made up of Referential Attributes in a Referring Participant Class.

The same Class may formalize multiple Associations, but a distinct Formalizing Class is created for each such Association. So while a Class itself may hold multiple References, there will only be one To One Reference for each formalization role it plays in a non-associative Association.