miUML Metamodel Class Descriptions
Relationship Subsystem

Leon Starr

Sunday, March 4, 2012

mint.miUMLmeta.td.3

Version 2.3.3

Table of Contents

Unary Association **UASSOC** 48

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] The OOA 96 Report, Project Technology, Sally Shlaer and Neil Davenport, 1996 <URL>

[HTBCM] How to Build Class Models, Leon Starr

Active Perspective APERS

A Binary Association has two Perspectives, one Active and one Passive Perspective.

In fact, the two sides of an Association could have just as easily been designated as the A side and the B side. Using the terms Active / Passive offers the modeler a systematic way to choose the phrase to apply to each side. For example, the phrase pair configures / is configured by readily establishes the Perspective sides.

If it's not clear from the phrase names which side should be active or passive, then arbitrarily assign each role and be done with it. You can always query the metamodel later to find out which is which. Any miUML class diagram editor should provide easy UI access to this query (highlight the A/P sides).

Attributes

Rnum

Asymmetric Perspective.Rnum and Binary Association.Rnum — See description of Domain attribute.

Domain

Asymmetric Perspective.Domain and Binary Association.Rnum — This Active Perspective is one side or the other of this Binary Association and is generalized as an Asymmetric Perspective.

Side - constrained

Asymmetric Perspective.Side — This attribute is necessary to compose a complete reference to the superclass. Constrained to the single value: active.

Identifiers

I> Rnum + Domain

Only one of each Perspective type (Active or Passive) is allowed on a Binary Association.

Relationships

R124

Binary Association IS VIEWED FROM *exactly one* Active Perspective
Active Perspective IS A VIEW ON *exactly one* Binary Association

On a Binary Association between aircraft and runway, let's say, the point of view from the aircraft might be 'will take off from'. Since we can easily state this point of view in the active

voice, it seems logical to establish it as the Active Perspective. From the opposite point of view, from the runway, the Perspective would then be Passive, regardless of how it is stated since the Active Perspective has already been assigned, such as 'is location for takeoff of'.

By definition, a Binary Association must have two Perspectives and, also by definition, one Perspective will be Active and the other Passive. The phrases may or may not be written in the corresponding active/passive voice.

An Active Perspective exists only as a side of a single Binary Association.

Association ASSOC

"An *association* is the abstraction of a set of domain relationships that hold systematically between different kinds of conceptual entities, abstracted as classes, in the domain." [MB]

miUML uses the definition in Executable UML above.

Attributes

Rnum

Relationship.Rnum — See description of Domain attribute.

Domain

Relationship.Domain — The corresponding superclass identifier component.

Identifiers

I> Rnum + Domain

Same as the superclass.

Relationships

R119
Association IS A Unary or Binary Association

A Unary Association has only one Perspective, whereas a Binary Association, as the name also implies, has two.

The Unary Association is introduced in reference [OOA96]. It captures a reflexive association on a single Class where there is only one Perspective and it would be nonsense to suggest that there are two distinct multiplicities. On a map, for example, you may say that one territory is adjacent to another territory. There is no direction on the concept of adjacency in this example. Consequently, we would just say that the multiplicity is many and the Perspective name is 'is adjacent to'.

UML does not explicitly support Unary Associations, but it does accommodate them. Both sides of an Association may have identical names and multiplicities. By contrast, miUML does explicitly support Unary Associations because the modeler's intent should be clear. and unambiguous.

Association Class ACL

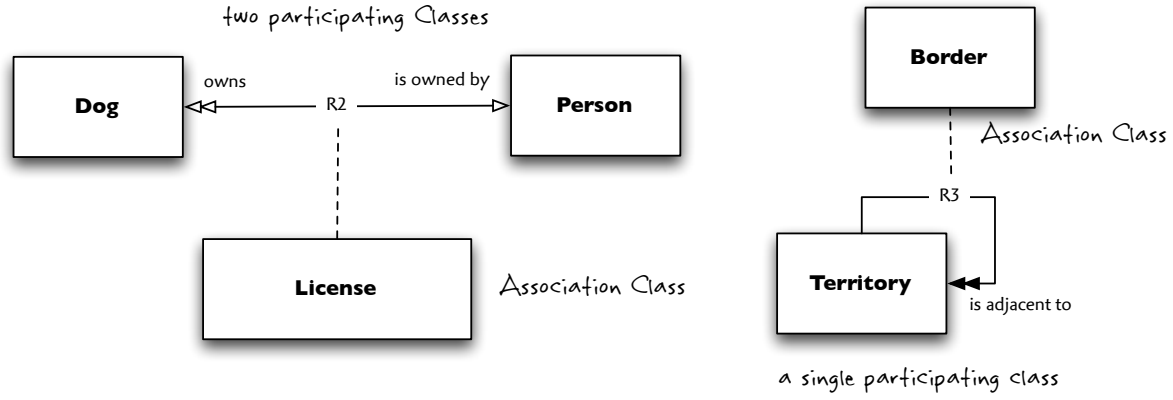
An *Association Class* is a Class that abstracts a set of things each of which is dependent on a Link on some Association. A Dog License, for example, may crystallize the owns Association between the Person and Dog Classes. Each Dog License Object would come into existence as a Link is established between a Person and a Dog on the owns Association. When a Dog License is deleted, the corresponding Link between Person and Dog would also be eliminated. Under no circumstances may a Dog License Object exist independently of a Person — owns — Dog Link.

To play the role of Association Class, a Class must be bound to a single Association, shown on the class diagram with a dashed UML dependency connector as illustrated below. The Referential Attributes and one or more Identifiers are composed according to well defined relational rules which take into account the multiplicity on each side of the bound Association. These rules are described in the Formalization Subsystem.

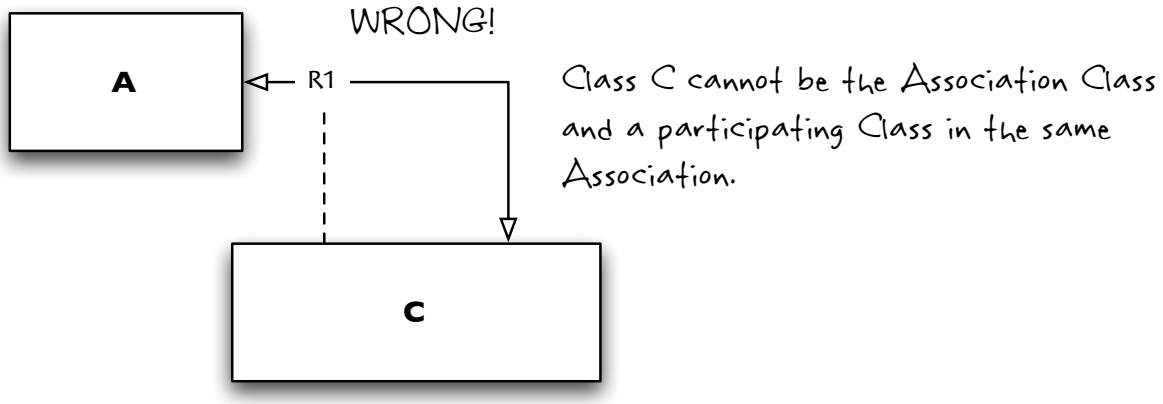
An Association bound to an Association Class consists of one or two participating classes and one Association Class. In the Dog License example, Dog and Person would play the roles of participating classes and Dog License would be the Association Class. If the Association is reflexive, there will be only one participating class and one Association Class.

As shown below, a Class may not play both the participating class role and the Association Class role on the same Association. Furthermore, a Class may not play the Association Class role on more than one Association. Other than those restrictions, any Specialized or Non-specialized Class may play the role of Association Class on some Association.

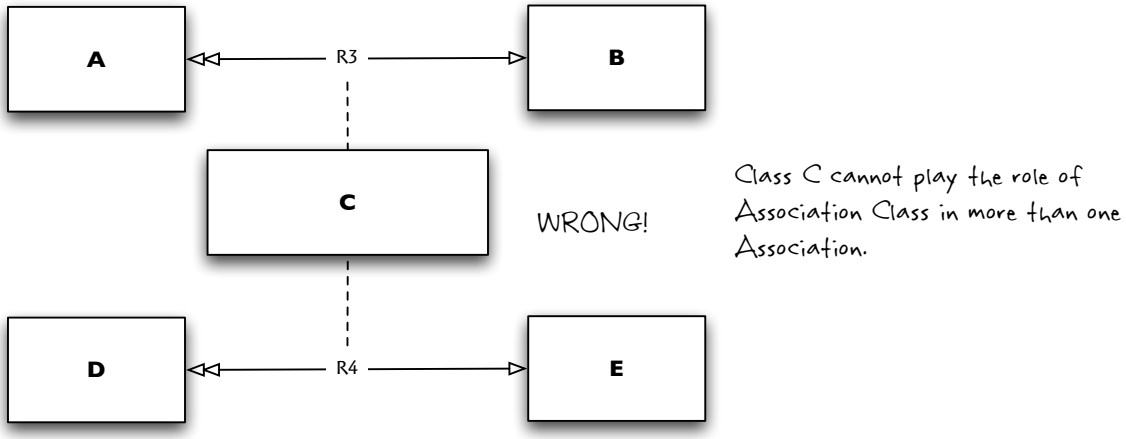
Association Class examples



Mutual exclusion of participating and Association Class roles.



A Class may play the Association Class role more than once



Attributes

Rnum - constrained

Association.Rnum — The Association formalized by the Class.

This attribute is constrained such that it may not assume the value of any other Relationship Rnum in which this Class participates.

Class

Class.Name — The Class playing the Association Class Role.

Domain

Association.Domain and Class.Domain — an Association Class is in the same Domain as the Association it formalizes.

Identifiers

1> Rnum + Domain

Standard 1x:1x association class identifier composition rules. Each participating class identifier must be an identifier of the association class.

2> Class + Domain

See note for 1> above.

Relationships

R120
Class ABSTRACTS DEPENDENCY ON *zero or one* Association
Association ABSTRACTED DEPENDENCY IS FULFILLED BY *zero or one* Class

A Class may play the role of an Association Class in any Association in which the Class does not already participate. In doing so, the Class establishes a dependency on the Association such that each object in the Association Class manifests a link on the Association on which it abstracts dependency.

Any given Class may or may not play the role of Association Class on some Association. Any given Association may or may not be supported by an Association Class. In fact, there are certain cases where, due to the rules for Referential Attribute placement, an Association Class will be required on certain Associations. These rules are captured in the Formalization Subsystem.

Asymmetric Perspective ASYMP

Each side of a Binary Association has a distinct Perspective, either Active or Passive. Since each side is from a different point of view, it establishes an *Asymmetric Perspective*.

For example, Folder contains File is not the same as File is contained within Folder. These are opposing and distinct (asymmetric) Perspectives.

Attributes

Side - constrained

Perspective.Side — Indicates one of two possible Perspectives on a Binary Association.

Constrained to values [active | passive]

Rnum

Perspective.Rnum

Domain

Perspective.Domain

Identifiers

1> Side + Rnum + Domain

An Asymmetric Perspective is on one side or the other of a Binary Association. It is not possible to have two Active or two Passive Perspectives on an Association.

Relationships

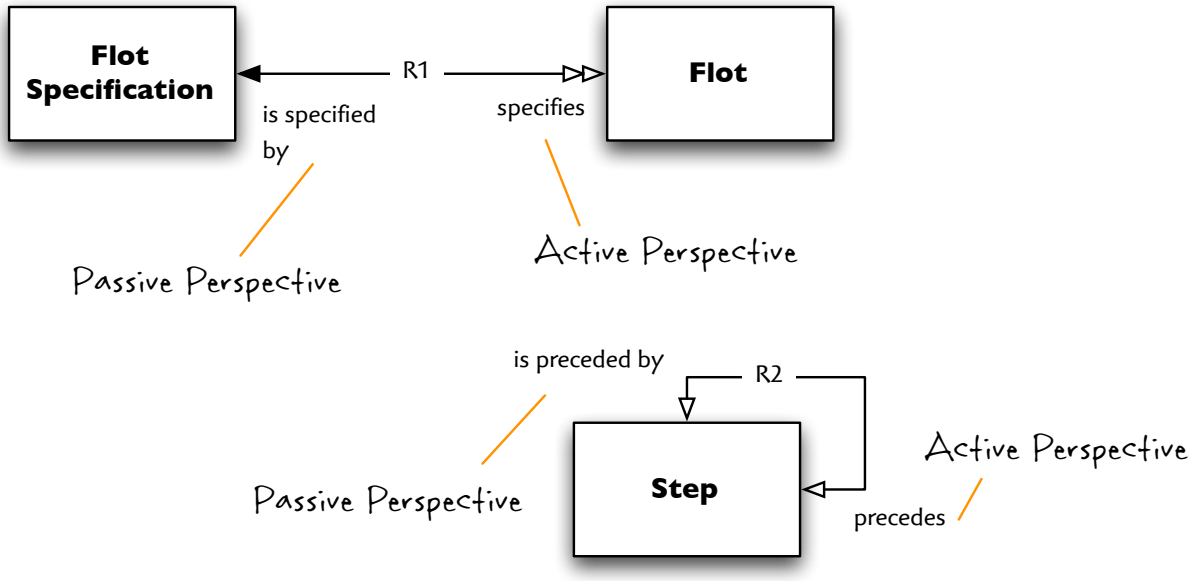
R105
Asymmetric Perspective IS AN Active or Passive Perspective

A Binary Association, by definition, has two Perspectives. By policy, each is either Active or Passive.

Binary Association BASSOC

The term 'binary' means that there are exactly two Perspectives on this type of Association. It does NOT mean that there are two Classes. A reflexive Binary Association may be created on a single Class such that each of the two Perspectives is viewed from the same Class.

Here are two examples of Binary Associations. One is drawn between two Classes and the other is reflexive on a single Class.



There are still two distinct Perspectives in the reflexive case even though the multiplicity and conditionality happen to be the same.

Had the modeler chosen the names is executed before / is executed after for the reflexive example, the active/passive designations would be applied arbitrarily since both names would be in the active voice.

Attributes

Rnum

Association.Rnum

Domain

Association.Domain

Identifiers

1> Rnum + Domain

Same as the superclass.

Relationships

None.

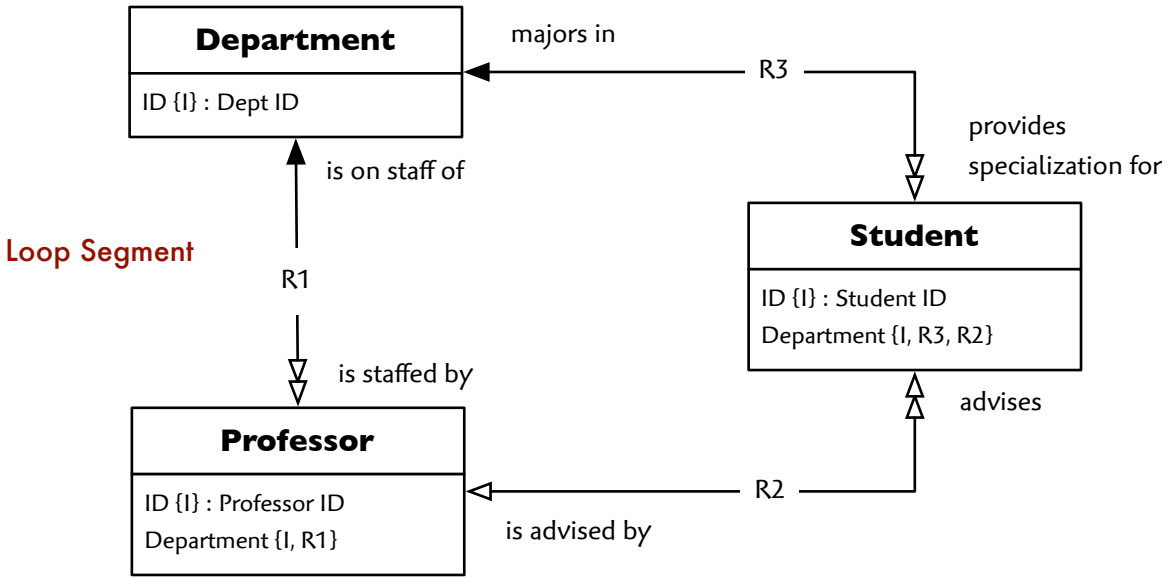
Constrained Loop CLOOP

A path of one or more Relationships that starts at some arbitrary origin Class and returns back to the same Class forms a loop. Starting at some arbitrary initial Class in the loop take one arbitrary initial instance and navigate one step in any direction, clockwise for example, to obtain a set of one or more instances (depending on the Relationship) which we will call instance set A from the target Class. Now starting with the same initial instance in the initial Class, navigate the opposite direction in the loop, counterclockwise in this example, back to the same target class to obtain instance set B. A *constrained loop* requires that B is necessarily a subset of A.

Constrained Loop

CL1: (R1, R2, R3)

A Constrained Loop is a set (not ordered) of contiguous Relationships.



By incorporating Department.ID into both Student and Professor identifiers and merging the Department referential attribute in Student, a Student must be advised by a Professor on the staff of the same Department as the Student's major.

See [HTBCM] for a a detailed illustration of the set constraints on the above example.

[MB] refers to the case where B is always an improper subset of A, in other words, B = A as an 'equal set constraint'. The case where B is always a subset, but not necessarily equal to A, is referred to in [MB] as a 'subset constraint'. Whether or not a an equivalence or a subset constraint applies is simply the consequence of the aggregate multiplicities present in the loop.

There is no need then to record the distinction in the metamodel as the statement that a loop is constrained is adequate to make the analysis intention clear.

A Relationship in a Constrained Loop may be either an Association or a Generalization.

No Relationship in a Constrained Loop is 'special'. This is an important point as the notation recommended in [MB] and [HTBCM] and [OOA96] to indicate a Constrained Loop can be mis-interpreted since the constraint is typically written next to a specific Association. Regardless of notation used, it is best to think of a Constrained Loop as simply a set of interconnected Relationships designated as constrained. In other words, the loop is the item of interest, not any particular component Relationship.

See the references mentioned above for specific examples of Constrained Loops.

Attributes

Element

Spanning Element.Element — From the superclass.

Domain

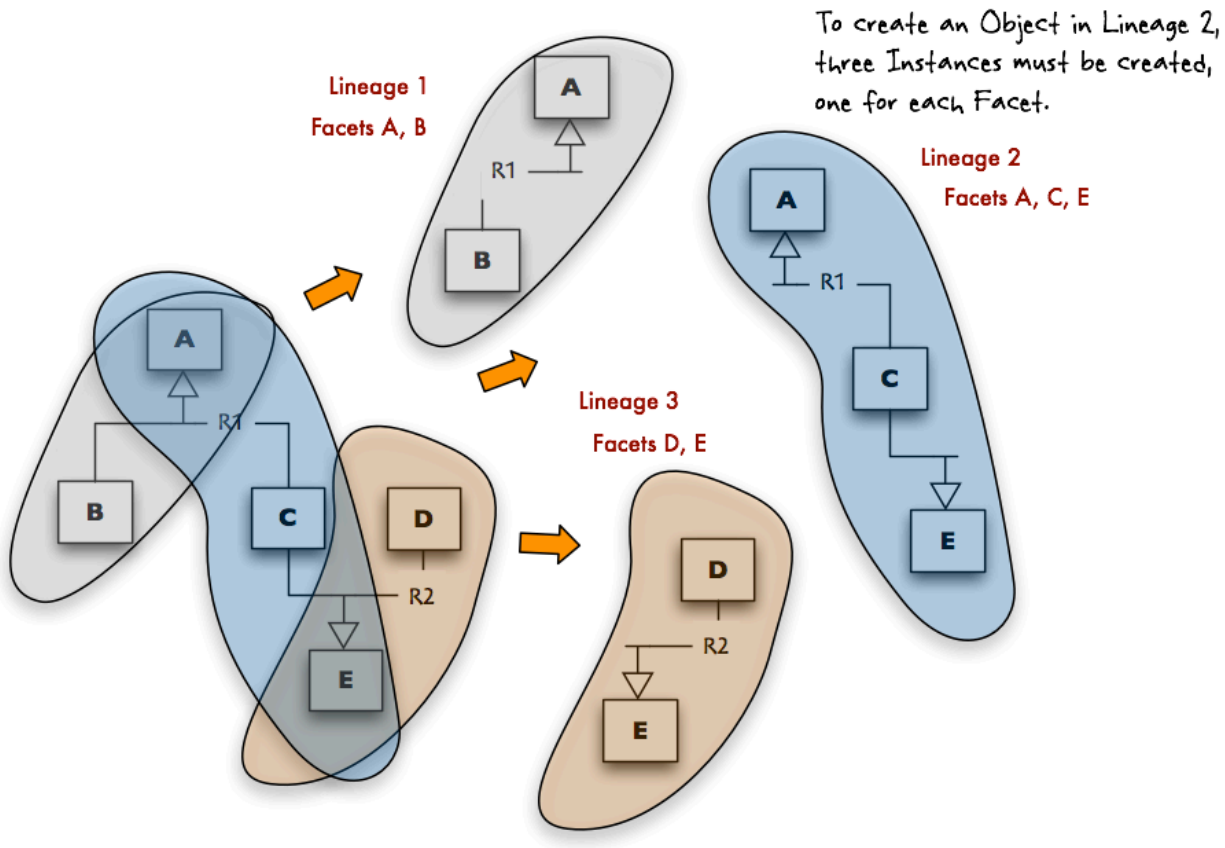
Spanning Element.Domain — From the superclass.

Relationships

None.

Facet FACET

An Object that participates in one or more interconnected Generalizations has many *facets*, each of which represents a required membership in a set abstracted by a Specialized Class in a Lineage. At creation, each Facet in a selected Lineage must be instantiated to yield a complete Object. For an existing Object, each Facet in the Object's Lineage indicates an Instance that must be deleted to completely remove the Object.



Attributes

Lineage

Lineage.Element — The Lineage requiring instantiation.

Class

Specialized Class.Name — The Specialized Class that must be instantiated.

Domain

Specialized Class.Domain and Lineage.Domain — The Specialized Class must be in the same Domain as the Lineage.

Identifiers

1> Lineage + Class + Domain

Standard Mx:Mx association class identifier composition.

Relationships

R131

Lineage DEFINES A FACET OF AN OBJECT WITH *one or many* Specialized Class
Specialized Class ABSTRACTS A FACET OF AN OBJECT WITHIN *one or many* Lineage

In the simplest case of a single Generalization, a Lineage will consist of a one Superclass and a one Subclass, with a single Lineage per subclass branch. Therefore, the minimum number of Specialized Classes defined in a Lineage is two.

By definition, a Specialized Class abstracts only part of a complete Object, and a Lineage defines all Facets required to define a complete Object. Consequently, each Specialized Class is a member of at least one Lineage.

The same Specialized Class may participate in more than one Lineage if the Specialized Class abstracts a subset that either can be subdivided into more subsets (modeled by Subclasses) or is included in more than one superset (modeled by Superclasses).

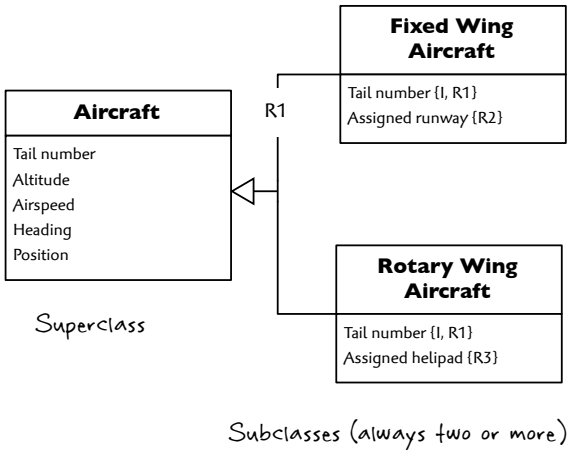
Generalization GEN

A miUML *Generalization* is a type of Relationship that abstracts the concept of mutually exclusive sets. It corresponds to a UML generalization with the standard tags { disjoint, complete }.

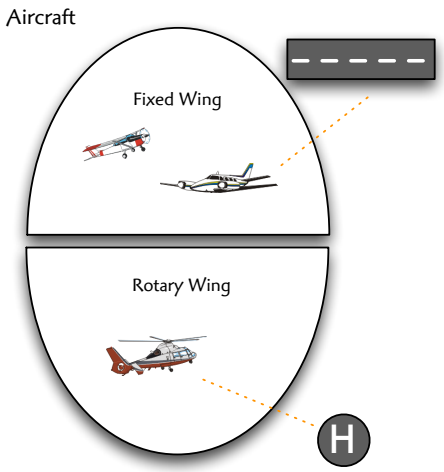
Generalization may be used when a proposed Class features Attributes, behavior, Relationships and policies common to all Objects, but also establishes Attributes, behavior, Relationship and policies specific to certain subgroups of Objects.

If we consider the set of all aircraft, for example, we might see Attributes such as altitude, airspeed, heading and so forth which apply to any aircraft. But certain aircraft, such as helicopters may land on a helipad while non-helicopters may only land on a runway. A Generalization Relationship, shown below, may be constructed which establishes a Superclass to represent aircraft in general and two subclasses, fixed wing aircraft and rotary wing aircraft to establish a partition on the set of all aircraft.

Example miUML Generalization Relationship



Abstracted set partitioning

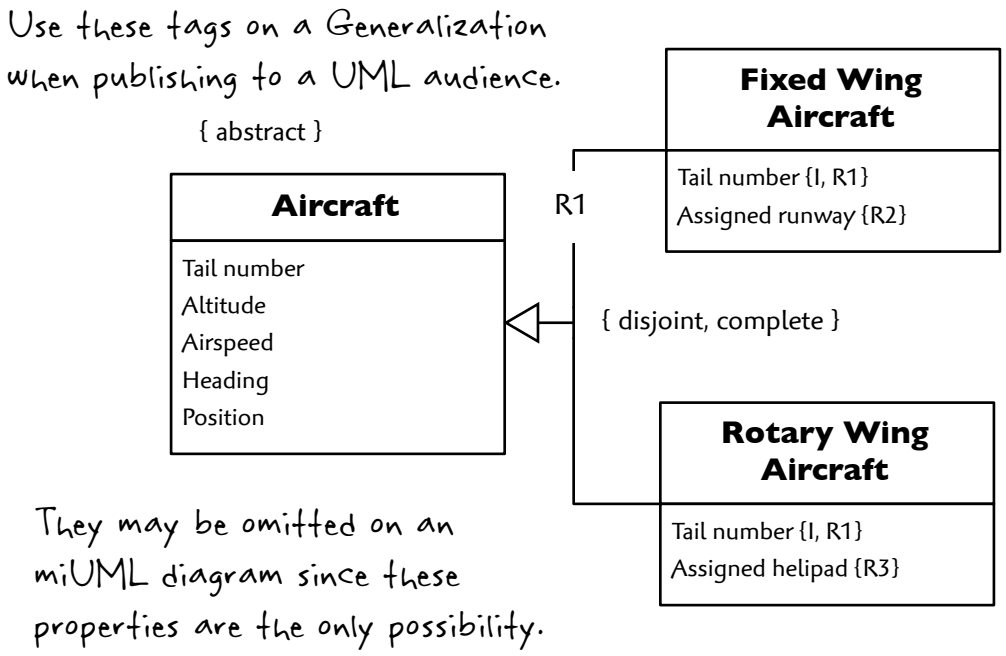


In a miUML Generalization, each object is represented by a single Superclass instance and a single Subclass instance. There is no such thing as an aircraft, according to the model shown above, which is not either a fixed wing aircraft or a rotary wing aircraft. Also, there is no such thing as an aircraft which is both, again, according to the model. Different requirements, accommodating VTOL aircraft, for example, may necessitate a different model and possibly a combination of multiple Generalizations and additional Associations.

Certain tags should be added to class diagrams when translated from miUML to OMG UML. Each Superclass should be tagged { abstract } and each Generalization should be tagged { dis-

joint, complete }. Since all Generalizations in miUML conform to these tags, they are unnecessary when it is understood that the class diagram is a miUML class diagram.

Export of Generalization from miUML to UML class diagram



Attributes

Rum

Relationship.Rnum and Superclass.Rnum — Both the Superclass and Generalization are part of the same Relationship.

Domain

Relationship.Domain — Both the Superclass and Generalization are part of the same Relationship and, hence, Domain.

Superclass

Superclass.Class — The Superclass of this Generalization.

Identifiers

I> Rnum + Domain

Must be the same as the Relationship superclass.

RELATIONSHIPS

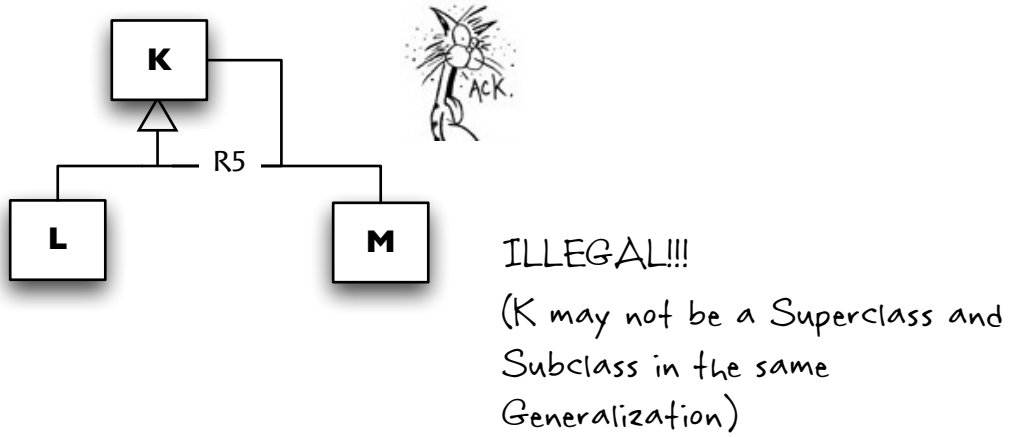
R103

Generalization REQUIRES *exactly one* Superclass
Superclass IS REQUIRED BY *exactly one* Generalization

By definition, a Generalization defines a single Superclass.

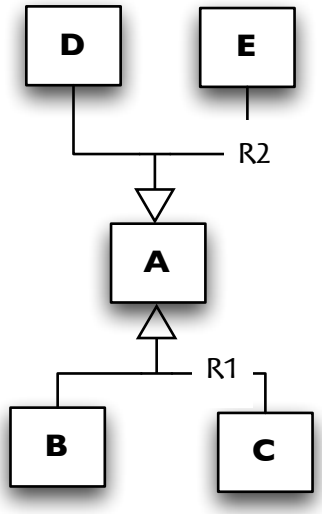
A Superclass represents the participation of a Class as the general case in a Generalization. A Class may participate in a Generalization as either a Superclass or a Subclass. In either case, a Class may not play more than one role in the same Generalization.

One role per Generalization



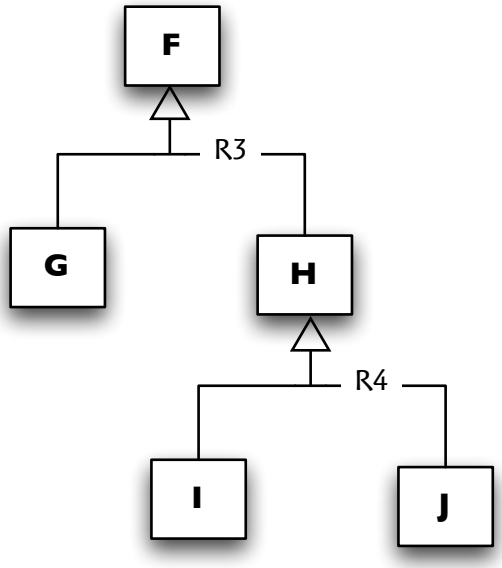
A Class may, however, play multiple roles, but only one per Generalization. So a Class may be simultaneously the Superclass in two or more Generalizations (multidirectional generalization), or the Superclass in one Generalization and a Subclass in another (multilevel generalization), for example.

Multidimensional Generalization



The Class A is a Superclass in R2 and also a Superclass in R1.

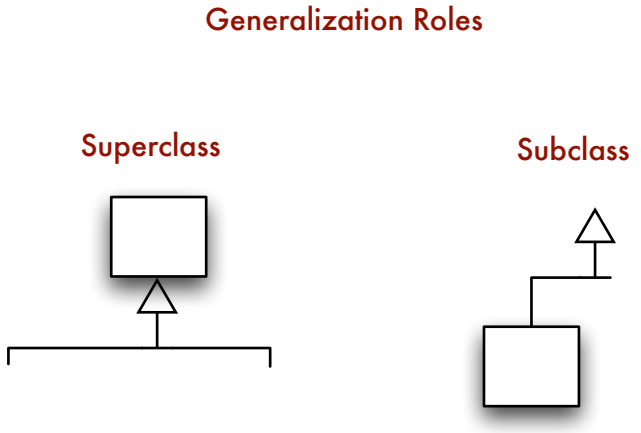
Multilevel Generalization



The Class H is a Subclass on R3 and also a Superclass on R4.

Generalization Role GROL

By definition, a Specialized Class participates in at least one Generalization. A Specialized Class's participation in a Generalization is a *generalization role*. The Superclass and Subclass roles are the only two possible Generalization Roles. In other words, each Class participating in a Generalization is either a Superclass or a Subclass. A Specialized Class may only play one role within a given Generalization.



Attributes

Rnum

Generalization.Rnum — The Specialized Class takes part in this Generalization.

Class

Specialized Class.Name — The Specialized Class taking part.

Domain

Specialized Class.Domain and Generalization.Domain — Both the Specialized Class and Generalization must be in the same Domain.

Identifiers

1> Rnum + Class + Domain

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

RELATIONSHIPS

R101

Specialized Class PARTICIPATES IN *one or many* Generalization
Generalization HAS PARTICIPATING *one or many* Specialized Class

By definition, a Specialized Class participates in at least one Generalization.

A Generalization requires one Superclass and two Subclasses at a minimum so there is, in fact, at least three Specialized Classes participating in any Generalization.

R102

Generalization Role IS A Subclass or Superclass

In a single Generalization, a Specialized Class participates as either a Superclass or a Subclass. The same Specialized Class may participate in many different Generalization Roles, but only one per Generalization.

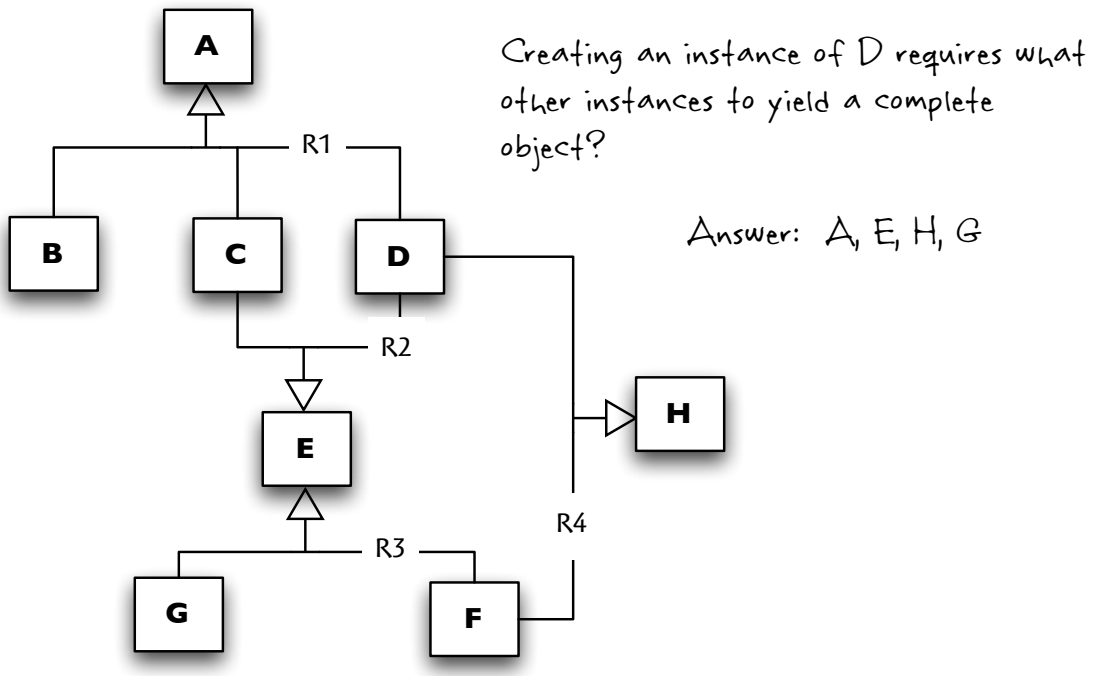
Lineage LIN

To create an Object that participates in one or more Generalizations, it is necessary to instantiate multiple Specialized Classes. In the simplest case, only one Generalization is involved requiring a single Superclass instance and a single Subclass instance. Incomplete objects are illegal in miUML, so it is always necessary to leave a complete Object or no Object at the conclusion of a runtime transaction. (It is up to the modeler to organize processing into adequately independent transactions). It is not okay, to create a Superclass instance and then omit the required Subclass instance or vice versa.

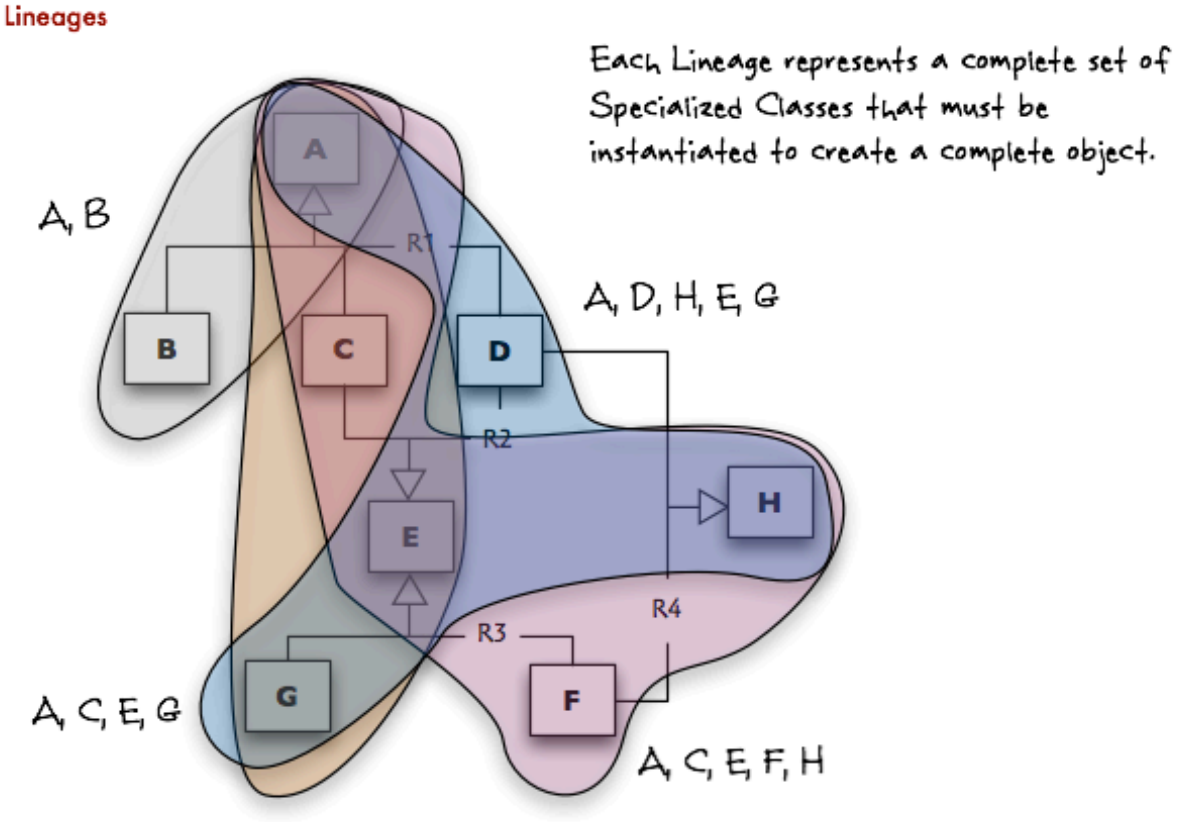
If an object can migrate to a different specialization (change its Subclass), it is necessary to ensure that enough new Instances are created to end up with a complete Object while ensuring that all all Instances pertinent only to the prior specialization are deleted. In the single Generalization case, this simply means creating one new Subclass instance and deleting one old Subclass instance.

It gets a bit more interesting when multiple Generalizations are interconnected as shown.

Instantiation in interconnected Generalization Relationships



In the example above, instantiation of D requires instantiation of the Specialized Classes A, E, H and G. Note that the R3 specialization is predetermined by the choice of D in R1. E must be specialized as G. Fortunately, a straightforward algorithm can be run at metamodel population time (model edit time) to determine which *lineage* of Specialized Classes must be instantiated or deleted as a group. Lineages for the above example are shown here:

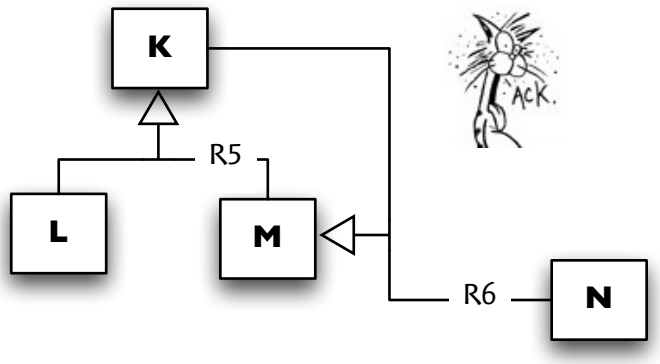


There are a few advantages to incorporating the Lineage concept into the miUML metamodel. It makes it possible to validate the integrity of Generalization data (M0) populations. A model editor can take advantage of this information to warn the model developer when action language yields an incomplete object. Also, the metamodel can be consulted during action language execution to verify correct instantiation.

Furthermore, the Lineage concept is essential when verifying that a newly created Subclass does not, through a chain of references end up as a Superclass of itself and vice versa.

Illegal Lineage

A cycle is illegal within a Lineage. In other words a Subclass may not be a Superclass of itself and a Superclass may not be a Subclass of itself.



And by incorporating a Lineage tracing algorithm into the miUML metamodel, there is no need for each model editor to reinvent a potentially faulty algorithm.

But perhaps the most important reason for modeling Lineage is to provide a precise meaning of 'object'. In short, an Object is either an instance of a Non-Specialized Class or a set of instances of Specialized Classes in a Lineage.

Attributes

Element

Spanning Element.Number — A Lineage may span more than one Subsystem, so it is a Spanning Element.

Domain

Spanning Element.Domain

Identifiers

1> Element + Domain

All Elements have the same identification scheme.

Relationships

None.

Loop Segment LSEG

A *loop segment* is a member Relationship in a Constrained Loop. More precisely, it is a *constrained* loop segment, but there is no need to model unconstrained loop segments, so the simpler name is unambiguous.

Attributes

Loop

Constrained Loop.Element —The Loop Segment is a component of this Constrained Loop.

Rnum

Relationship.Rnum —The Relationship member.

Domain

Relationship.Domain and Constrained Loop.Domain — Each is in the same Domain.

Identifiers

1> Loop + Rnum + Domain

Standard Mx:Mx association class identifier composition.

RELATIONSHIPS

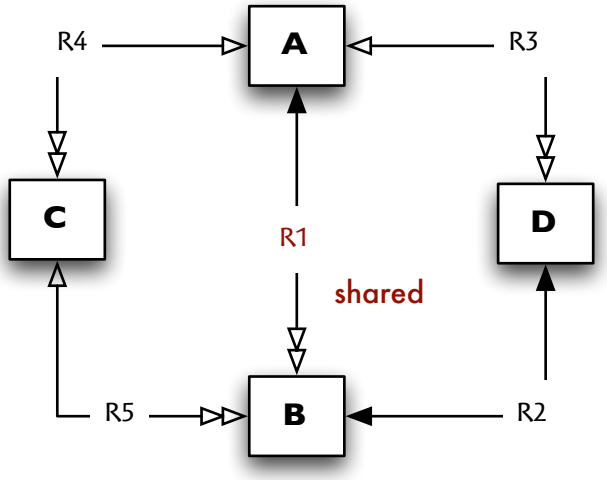
R160

Relationship IS A SEGMENT OF *zero, one or many* Constrained Loop
Constrained Loop IS A CONTIGUOUS CLOSED PATH OF *one or many* Relationship

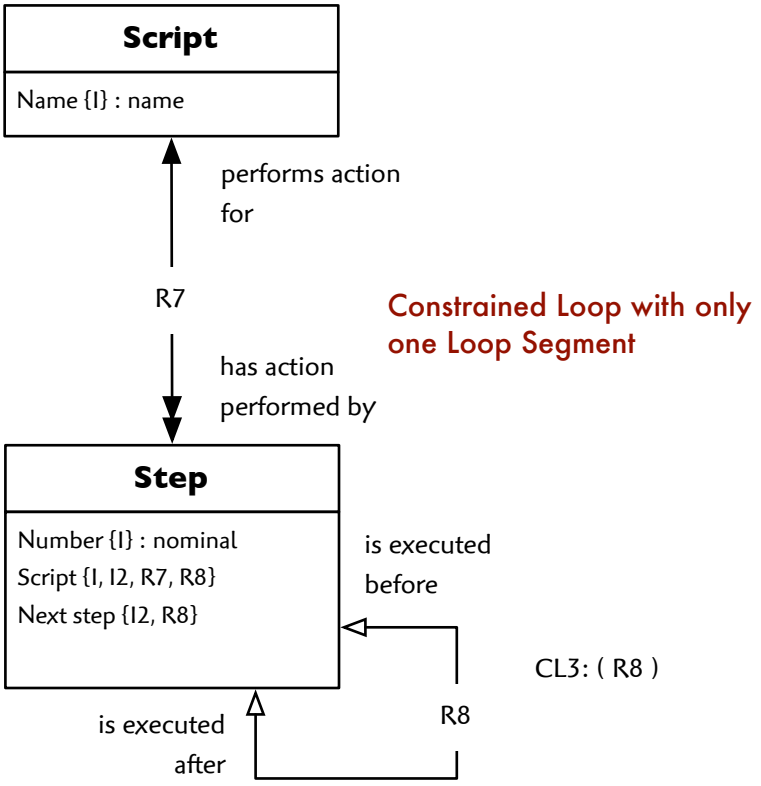
A Relationship may or may not participate in any Constrained Loops. As shown below, the same Relationship may be part of multiple Constrained Loops.

A shared Loop Segment

CL1: (R1, R2, R3)
CL2: (R1, R4, R5)



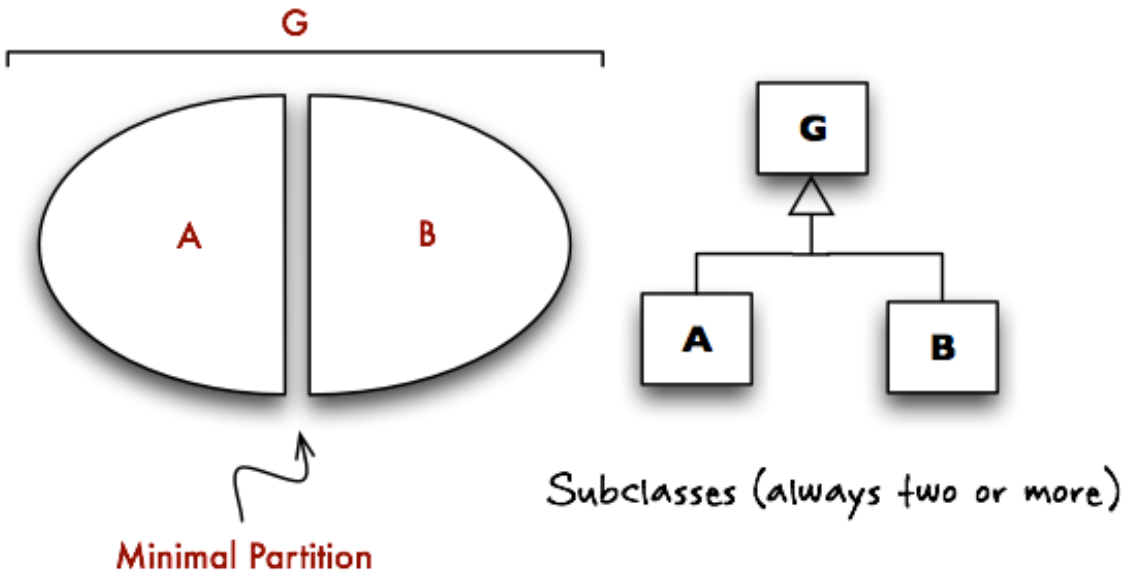
A Constrained Loop, by definition, consists of at least one Relationship. A single constrained Reflexive Association, as shown, constitutes a minimal Constrained Loop.



Since a Step may only follow another Step within the same Script, R8 constitutes a Constrained Loop.

Minimal Partition MPART

A miUML Generalization abstracts a general set that has been partitioned at least once to yield two proper subsets. Each additional partition yields one more subset. The general set is abstracted as a Superclass and each proper subset is abstracted as a Subclass.



As illustrated above, the set abstracted by a Superclass requires a *minimal partition* to yield the required minimum of two Subclasses. The Subclasses on either side of this Minimal Partition are arbitrarily designated A and B.

This constraint has been formalized to preclude the abuse, in Executable UML, of modeling a Generalization with only one Subclass. In set theory, an improper subset is a subset that includes the entire superset. This sounds reasonable enough, but what does it mean in terms of object oriented analysis? Once a Class is defined, why abstract it again? The chief abuse is to evade the rule that a Class has one lIfecycle. Abstract a Class twice and get two lifecycles. But the one-lifecycle rule is the foundation of Class definition in miUML, so the need for two distinct lifecycles implies that the analysis is flawed and that there are in fact two distinct Classes required (or the subject matter Domain is abstracted poorly or any number of possible analysis errors). Another possible motivation for redundant Class definition is to work around some essential feature lacking in a vendor's draw tool or model compiler. A key goal of miUML is to jettison tool dependent and other legacy hacks.

The history of programming languages has repeatedly demonstrated that the best intentioned language cannot police itself against clever abuse. That's always going to happen, and not always for the worse. On the other hand, there is no reason to provide features ripe for mischief that have no other utility!

Attributes

Rnum

The Rnum of the Subclass (Subclass.Rnum) on each side of the Minimal Partition and the Rnum of the Generalization (Generalization.Rnum)

Domain

The domain of the Subclass (Subclass.Domain) on each side of the Minimal Partition and the domain of the Generalization (Generalization.Domain)

A subclass

Subclass.Class — This is the name of the Class on the arbitrarily named 'A' side of the Minimal Partition.

B subclass - constrained

Subclass.Class — This is the name of the Class on the arbitrarily named 'B' side of the Minimal Partition.

This value is constrained so that it does not equal the value of the A subclass attribute.

Identifiers

1> Rnum + Domain

Each Generalization has exactly one Minimal Partition.

R116

Generalization REQUIRES *exactly one* Minimal Partition
Minimal Partition IS REQUIRED BY *exactly one* Generalization

A Generalization is always split across at least two Subclasses with a Minimal Partition. This is just the required minimum. Generalization may be partitioned multiple times resulting in S-1 partitions where S is the number of Subclasses. But only the minimal constraint is relevant, so only one required Minimal Partition is modeled per Generalization.

R117

Minimal Partition YIELDS AS A *exactly one* Subclass
Subclass IS YIELDED ON A SIDE OF *zero or one* Minimal Partition

A Minimal Partition splits the set abstracted by the Superclass into two proper subsets arbitrarily designated as A and B, each of which corresponds to some Subclass in the same Generalization.

A Subclass may be on the A side of the Minimal Partition, on the B side or neither. If neither, then there must be at least three Subclasses in the Generalization.

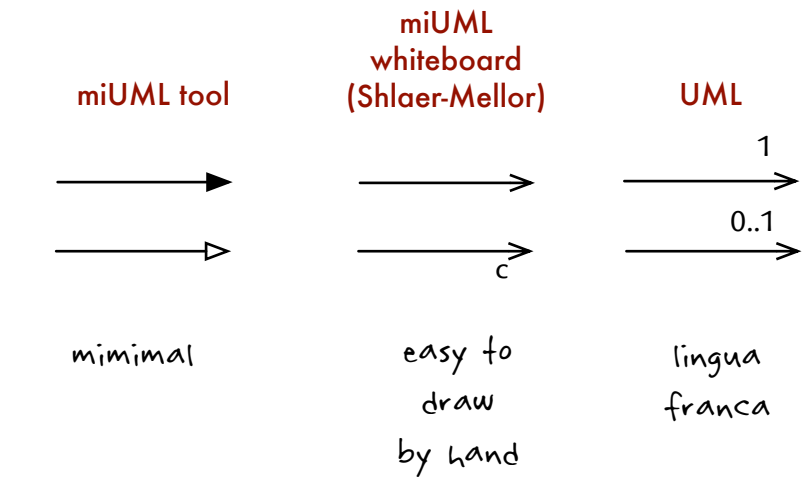
R118

Minimal Partition YIELDS AS B *exactly one* Subclass
Subclass IS YIELDED ON A SIDE OF *zero or one* Minimal Partition

See description for 'yields as A', above.

One Perspective ONE

This is a Perspective with a multiplicity of one.



Attributes

Rnum

Perspective.Rnum

Domain

Perspective.Domain

Side

Perspective.Side

Identifiers

1> Rnum + Domain + Side

Same as superclass.

Relationships

None.

Passive Perspective PPERS

See the description of Active Perspective. This is just the flip side.

Attributes

Rnum

Asymmetric Perspective.Rnum and Binary Association.Rnum — See description of Domain attribute.

Domain

Asymmetric Perspective.Domain and Binary Association.Rnum — This Passive Perspective is one side or the other of this Binary Association and is generalized as an Asymmetric Perspective.

Side - constrained

This attribute is necessary to compose a complete Reference to the Superclass. Always constrained to be the value: passive

Identifiers

1> Rnum + Domain

There is exactly one of each Perspective type (Active or Passive) on a Binary Association.

Relationships

R125
Binary Association IS VIEWED FROM *exactly one* Passive Perspective
Passive Perspective IS A VIEW ON *exactly one* Binary Association

See the description of the Active Perspective / Binary Association. This is just the flip side of that explanation.

Perspective PERS

A *Perspective* is a point of view from a hypothetical Instance on an Association.

Attributes

Side

The Perspective is on this side of an Association.

Type: [symmetric | active | passive]

Rnum

The Perspective is on this Relationship, which must be an Association. And, no, there is no need for an additional association between Perspective and Association to establish this constraint as it is already handled by (R121+R123+R119 or R121+R105+R124/5 + R119).

Type: Nominal

Domain

Class.Domain — Must be the same Domain as that of the Class to which this Perspective is attached.

Viewed class

Class.Name — The Perspective is observed from this Class.

Phrase

This is the verb phrase written on one side of an Association. It is the modeler's responsibility to ensure that the phrase is adequately precise, descriptive and correct with respect to the specified conditionality and multiplicity.

Type: Name

Conditional

True means that 0 is a possible multiplicity on this Perspective 0..1 or 0..*.

Type: Boolean

\ UML multiplicity

A UML multiplicity phrase, derived from the One/Many subclass and Conditional attributes.

Type: [0..1 | 0..* | 1 | 1..*]

RELATIONSHIPS

R104

Perspective IS A One or Many Perspective

The direction of reference of a Referential Attribute is determined by the availability of a One Perspective. Conditionality is less significant in this regard. So to capture the basic reference rule that 'it is always possible to refer to a One Perspective' it is necessary to abstract the One / Many specialization. See the Formalization Subsystem to see how it is used.

Regardless of conditionality, every Perspective is either One or Many (1, M) or, in UML terminology, (0..1, 1 or 0..*, 1..*).

R110

Perspective IS TAKEN FROM *exactly one* Class
Class IS STANDPOINT OF *zero, one or many* Perspective

A Perspective represents the point of view of an arbitrary Instance in some Class toward an Object linked along and on the other side of the Perpective's Association. If the Association is reflexive, the Object on the other side of the Link may, in fact, be the very same Object.

On a class diagram, each side of a Binary Association is labeled with a phrase, multiplicity and conditionality characterizing a single Perspective. A Unary Association will have only one Perspective labeled, but it should be understood that each Object on a Unary Association link does have a view toward the opposite Object (which may be itself). On a class diagram it is evident that a Perspective is always anchored on a single Class.

A Class that does not participate in any Association, or a Class that participates exclusively in the role of an Association Class will not have any attached Perspectives.

R121

Perspective IS A Symmetric or Asymmetric Perspective

A Perspective is either on a Unary Association, in which case it is Symmetric or it is on a Binary Association in which case it is Asymmetric.

Relationship REL

In miUML the term *relationship* describes time and instance invariant rules and policies that bind one or more Classes together in a Domain. The concept of 'time invariance', we means that the rules apply at all times. So a Departing Aircraft is directed to take-off from exactly one Runway, it is the case, at all times, that an Instance of Departing Aircraft is linked to exactly one Runway along the previously described Relationship. By 'instance invariant' we mean that the rules abstracted by a Relationship apply universally to all instances of the participating Classes.

Relationships are depicted as lines interconnecting the various Classes that appear on a class diagram. In miUML all Relationships are either Associations or Generalizations.

Attributes

Rnum

By policy, each Relationship is numbered uniquely within the scope of a Domain.

Type: Nominal

Domain

Subsystem Element.Domain

Element

Subsystem Element.Number — A Relationship is a type of Element.

Identifiers

1> Rnum + Domain

This is the intuitively appealing identifier which establishes the policy stated in the Rnum description above.

2> Element + Domain

Since all Elements are numbered uniquely within a Domain, and since a Relationship is an Element, this also serves as a distinct identifier.

Relationships

R100

Relationship IS A Generalization or an Association

The only UML Relationships relevant to an miUML class model are Generalizations and Associations. Each of these abstracts time invariant constraints that bind one or more Classes together.

Specialized Class SP_CLASS

Any Class that participates in at least one Generalization Relationship as either a Superclass or a Subclass is a *specialized class*.

An instance of a Specialized Class represents only one of multiple set memberships, abstracted by one or more other Specialized Classes, necessary to manifest a complete Object. So an instance of a Specification Class is never, itself, a complete Object.

Attributes

Name

Class.Name

Domain

Class.Domain

Identifiers

1> Name + Domain

Same as the superclass.

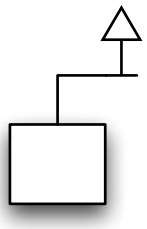
Relationships

None.

Subclass SUBC

Each of the proper subsets abstracted within a Generalization is a *subclass*. On the class diagram, a Subclass appears as a leaf in the Generalization Relationship hierarchy.

Subclass



Attributes

Rnum

Generalization Role.Rnum and Formalizing Class.Rnum

Class

Generalization Role.Class and Formalizing Class.Rnum

Domain

Generalization Role.Domain and Formalizing Class.Domain

Identifiers

1> Rnum + Class + Domain

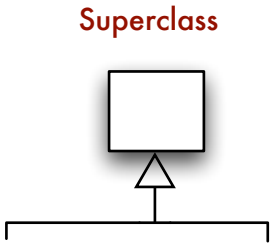
Same as superclasses.

Relationships

None.

Superclass SUPC

The general case superset abstracted for a Generalization is a *superclass*. On the class diagram, a Superclass appears as the root in the Generalization Relationship hierarchy.



Attributes

Rnum

Generalization Role.Rnum and Formalizing Class.Rnum

Class

Generalization Role.Class and Formalizing Class.Rnum

Domain

Generalization Role.Domain and Formalizing Class.Domain

Identifiers

1> Rnum + Class + Domain

Same as superclasses.

Relationships

None.

Symmetric Perspective SYMP

A Unary Association has only one Perspective. GIven two Objects (or the same Object linked to itself) on a Unary Association, the role played by either side of the Link is identical. There is, consequently, just one Symmetric Perspective.

For example, Territory borders Territory means the same thing for Territories A and B regardless of direction A->B or B->A. So A borders B means exactly the same as B borders A.

Therefore, only one phrase name, one multiplicity and one conditionality need be specified for a Unary Association.

Attributes

Rnum

Perspective.Rnum

Domain

Perspective.Domain

Side - constrained

Perspective.Side — Required to compose a complete reference to the Perspective superclass.

Constrained to the value: symmetric

Identifiers

1> Rnum + Domain

A Symmetric Perspective is on exactly one Unary Association so all you need is the Association / Relationship identifier.

Relationships

R123
Unary Association IS VIEWED FROM *exactly one* Symmetric Perspective
Symmetric Perspective IS A VIEW ON *exactly one* Unary Association

By definition, there is only one point of view on a Unary Association.

A Symmetric Perspective exists as the only side of a Unary Association.

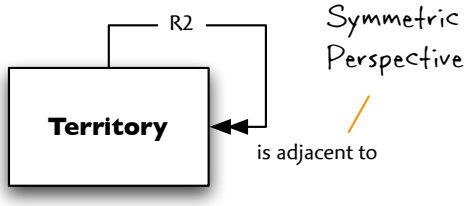
Unary Association UASSOC

The *Unary Association* models the 'symmetric reflexive relationship' described in [OOA96].

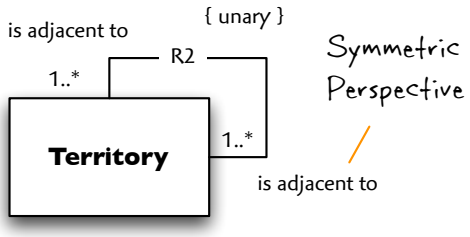
The term 'unary' means that there is only one Perspective on this type of Association. As a consequence, all Unary Associations are reflexive (on a single Class). A reflexive (single Class) Association is not necessarily Unary, however. So it is the number of Perspectives that matters.

Here is an example of a Unary Association:

miUML



Standard UML

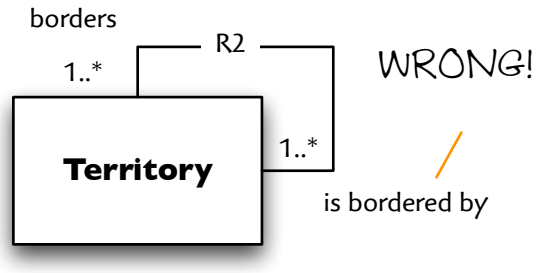


The other dummy perspective is created automatically when the diagram is generated from a miUML model. A tag should be also be added to clarify.

This example models a board game were there are no island territories, so every territory has at least one neighbor. The idea of 'adjacency' has no implied direction. Territory A is either adjacent to Territory B or it isn't. Put differently, saying that 'Territory A is adjacent to Territory B is exactly the same statement as Territory B is adjacent to Territory A'.

A reflexive Binary Association might be is attacking / is attacked by. Here there are two distinct Perspectives because the statement 'Territory A is attacking Territory B' is not the same fact as the statement 'Territory B is attacking Territory A'.

It is important for this aspect of reality to be captured precisely by the modeler. As shown above, UML treats all reflexive Associations as Binary, necessitating a redundant duplicate Perspective on one side. This leads to potential analysis errors such as the following where we slightly rename the Perspective in an active and passive voice even though the meaning is still identical:



The miUML meta model helps avoid this error by providing explicit support for Unary Associations.

When converting to UML notation, mirror the single, Symmetric Perspective (name, multiplicity and conditionality) and add a { unary } tag.

Watch out for redundant Links!

According to [OOA96] symmetric reflexive associations (Unary Associations) always require an Association Class regardless of multiplicity. The stated intention is to prevent more than one instance of the Association from being populated. This is a critical goal, but it is not entirely clear how the use of an Association Class alone would accomplish it other than by attracting more visual attention to the class diagram. So this rule is not enforced in the miUML meta-model. In either case, a constraint must be applied to preclude redundant links such as 'Territory A is adjacent to Territory B' and 'Territory B is adjacent to Territory A'.

Attributes

Rnum

Association.Rnum

Domain

Association.Domain

Identifiers

1> Rnum + Domain

Same as the superclass.

Relationships

None.