miUML Metamodel Class Descriptions
Classes and Attributes Subsystem

Leon Starr

Sunday, March 4, 2012

mint.miUMLmeta.td.1

Version 2.4.5

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

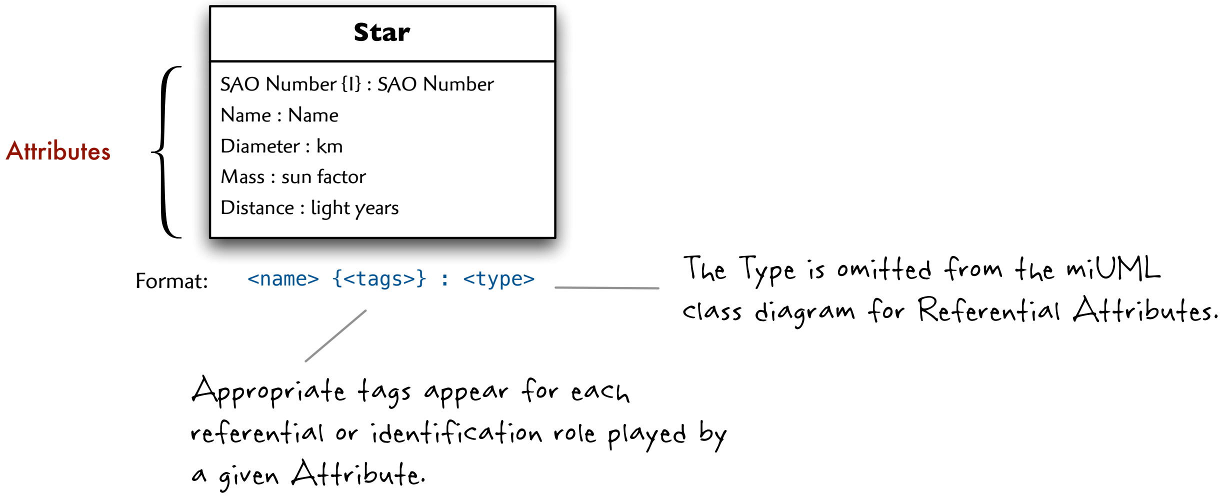
[SM1] Object-Oriented Systems Analysis, Modeling the World in Data, Sally Shlaer, Stephen J. Mellor, Yourdon Press, 1988, ISBN 0-13-629023-X

[HTBCM] Executable UML: How to Build Class Models, Leon Starr, Prentice-Hall, 2002, ISBN 0-13-067479-6

Attribute ATTR

"An *attribute* is the abstraction of a *single* characteristic possessed by all the entities that were, themselves, abstracted as a Class." [MB].

miUML uses the definition in Executable UML above.


Star
SAO Number {I} : SAO Number
Name : Name
Diameter : km
Mass : sun factor
Distance : light years
Attributes
Format: <name> {<tags>} : <type>
The Type is omitted from the miUML class diagram for Referential Attributes.
Appropriate tags appear for each referential or identification role played by a given Attribute.

The complete set of Attributes in a Class should be meaningful for all instances of that Class. This means that each instance of a Class should provide a value for each of its Attributes under all circumstances. The non-value 'not applicable' may never be used.

That said, there is a bit of an exception to this rule in the case of a Referential Attribute that refers to a 0..1 / 1c / one-conditional side of an Association. In this case, the Referential Attribute may be assigned the special value 'none' to indicate that no link is present at the moment. Note the difference with 'not applicable' in that the Referential Attribute is still applicable, but is not linked at the moment. It is similar in the sense that conditional references yield conditional logic (if the link is there, do this, if not do that) which is nice to avoid when possible for the same reason 'not applicable' is bad.

Attributes

Name

This is the name that appears inside the class rectangle symbol on a class diagram. Naturally, it may appear many other places. It should be descriptive and typically between 1-30 characters.

Type: Name

Class

Class.Name

Domain

Class.Domain

Identifiers

I> Name + Class + Domain

No two Attributes share the same name within a Class.

Relationships

R20
Attribute CHARACTERIZES *exactly one* Class
Class IS CHARACTERIZED BY *one or many* Attribute

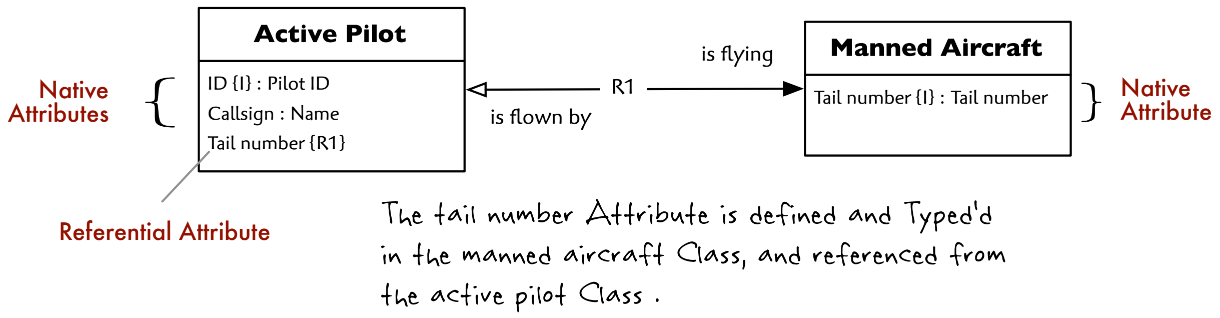
An Attribute expresses a characteristic that describes all potential instances of a Class.

Since a Class must have at least one Identifier, and an Identifier consists of at least one Attribute, it follows that a Class requires at least one Attribute.

An Attribute may not be defined outside of the context of a Class.

R21
Attribute IS A Native Attribute OR A Referential Attribute

An Attribute either refers to an Identifier Attribute in another Class or it does not. If it does, it is a Referential Attribute. Otherwise, it is a Native Attribute (native to its Class).

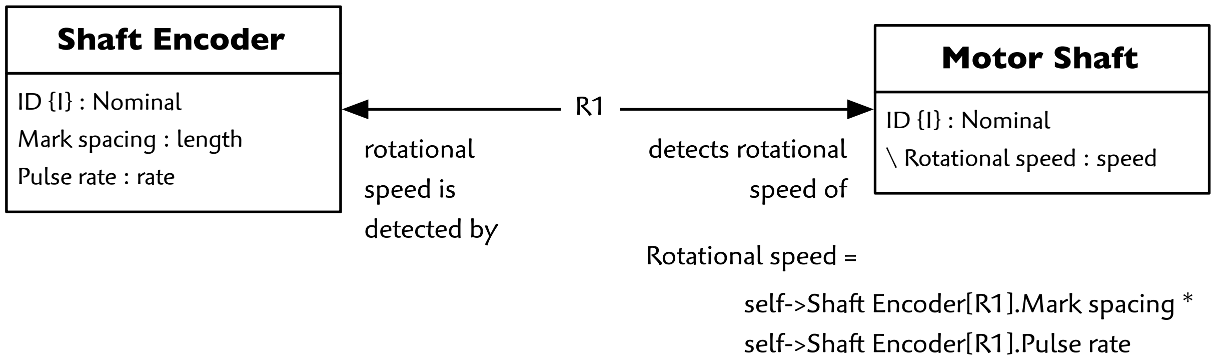

Active Pilot
ID {I} : Pilot ID
Callsign : Name
Tail number {R1}
Native Attributes
Referential Attribute
is flying
R1
is flown by
Manned Aircraft
Tail number {I} : Tail number
Native Attribute
The tail number Attribute is defined and Typed'd in the manned aircraft Class, and referenced from the active pilot Class.

Attribute in Derivation ATTR_DERIV

This is an Attribute that participates in a formula to compute the value of a Derived Attribute. The specific role in the Formula is not captured as a Formula specification language has not yet been formalized in miUML.

Attributes in Derivation

Shaft Encoder.Mark spacing -> Motor Shaft.Rotational speed
Shaft Encoder.Pulse rate -> Motor Shaft.Rotational speed


Shaft Encoder
ID {I} : Nominal
Mark spacing : length
Pulse rate : rate
R1
rotational speed is detected by
detects rotational speed of
Motor Shaft
ID {I} : Nominal
\ Rotational speed : speed
Rotational speed =
self->Shaft Encoder[R1].Mark spacing *
self->Shaft Encoder[R1].Pulse rate

Attributes

Source attribute - constrained

Attribute.Name — The Attribute contributing to a derivation. The value pair of Source attribute + Source class may not match the value pair of Target attribute + Target class. In other words, an Attribute may not be derived from itself.

Source class - constrained

Attribute.Class — Of the Attribute contributing to a derivation. See Source attribute above.

Target attribute

Derived Attribute.Name — The value of this Attribute is derived, in part, from the Source attribute.

Target class

Derived Attribute.Class — Of the Attribute that is derived.

Domain

Attribute.Domain and Derived Attribute.Domain — The Domain of both Source and Target Attributes.

Identifiers

I> Source attribute + Source class + Target attribute + Target class + Domain

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

Relationships

R26
Derived Attribute IS DERIVED FROM *one or many* Attribute
Attribute CONTRIBUTES TO THE VALUE OF *zero, one or many* Derived Attribute

By definition, a Derived Attribute must be computed from at least one other Attribute.

An Attribute of any type, including Derived, may contribute to the computation of a Derived Attribute (other than itself).

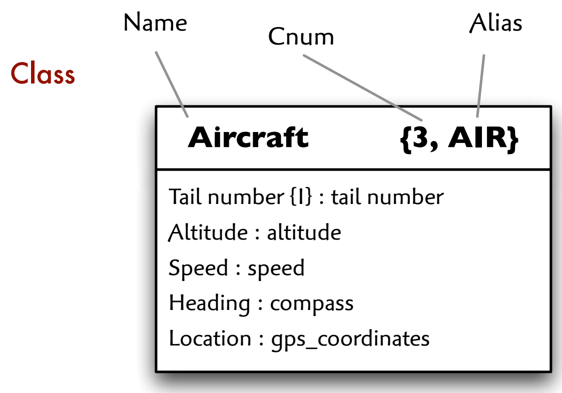
The specific role, or roles played by an Attribute in Derivation is determined by the Formula specified for the Derived Attribute.

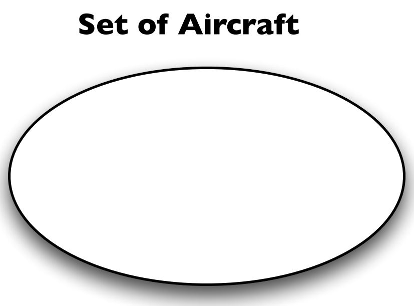
Class

"A *class* is an abstraction from a set of conceptual entities in a domain so that all the conceptual entities in the set have the same characteristics, and they are all subject to and behave according to the same rules and policies." [MB]

miUML uses the definition in Executable UML above.

A Class is represented symbolically on a miUML class diagram as a rectangle with separate rectangular compartments for a name, attributes and other model properties. *Ceci n'est pas une class*. This symbol is only a visual representation of a Class, and is not the Class itself. Clearly, a Class may be represented many other ways, such as textually.





The aircraft class is the definition of a set that may be populated with zero or more elements, each abstracted as an instance of the aircraft class.

In miUML a Class is just a set definition. It's not a Java class or a C++ class or an SQL table or any particular data structure. It could, however, be implemented as any of these.

Attributes

Name

This is the name that appears inside the top compartment of a class rectangle on a class diagram. Naturally, the name may appear many other places. It should be descriptive relative to the domain subject matter.

Type: Name

Cnum

A number, unique within a Domain, assigned to a Class for partial identification purposes.

Type: Nominal

Alias

A typically shorter name useful for referring to instances of a Class in a simulation, naming Events on a state chart and any other situation where an abbreviated label is desirable. For example:

```
AIR13: Request clearance( N31809Q )
```

An alias is optional since some developers may wish to just use full class names universally, or most of the time, and not bother with devising a set of aliases. Here the full name is used instead:

```
Aircraft13: Request clearance( N31809Q )
```

Consequently, this attribute cannot take part in a uniqueness constraint since many Alias values may be blank (and, hence, duplicate) in any given metamodel population. This also means that where Aliases are employed, there is the possibility of seeing duplicate labels: X3 defining an Event on a Class named Flot and X3 also referring to an Event on another Class named "Fragmetizer". (A lazy modeler may have used X as an Alias default). Consider Aliases as purely ornamental. The uniqueness of the two aforementioned Events would be established using fully qualified Class names such as Flot.X3.

Type: Short Name

Domain

Subsystem Element.Domain — This Class is a Subsystem Element in this Domain.

Element

Subsystem Element.Number — The Subsystem Element number of this Class.

Identifiers

1> Domain + Name

Each Class has a unique name within a Domain.

2> Domain + Element

Elements are all numbered uniquely within a Domain.

3> Domain + Cnum

Each Class is numbered uniquely within a Domain.

Relationships

R23
Class IS A Specialized or a Non-Specialized Class

If a Class participates in at least one Generalization, it is a Specialized Class. Otherwise, it is a Non-Specialized Class. A Specialized Class may play the role of Superclass or Subclass in each Generalization in which it participates. A Non-Specialized Class may not be a Superclass or Subclass in any Generalization.

The distinction is relevant in the interpretation of a Class instance. While an instance of a Non-Specialized Class is always a complete Object, an instance of a Specialized Class represents only some Facet of an Object.

Derived Attribute DV_ATTR

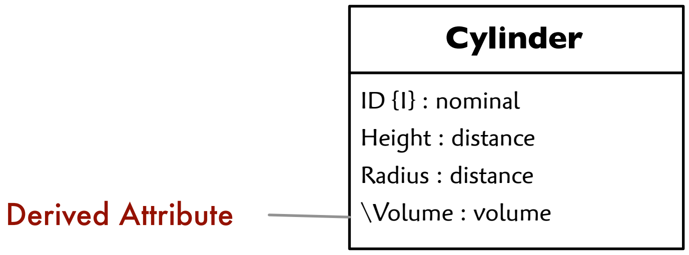
“A *derived attribute* is an Attribute whose value can be computed from other attributes already in the model.” [MB]

miUML uses and extends the definition in Executable UML above.

To be more specific, a Derived Attribute is computed from other Attributes within the same Domain in a state independent manner in a given slice of time. The first part goes without saying since Attributes in external Domains are always invisible. But the second part is less obvious. It means that all processing necessary to compute the current value of a Derived Attribute happens without the activation of any state transitions. To use one or more state transitions, as is often helpful, to compute an updated value of some Attribute, define that Attribute as an Independent Attribute instead.

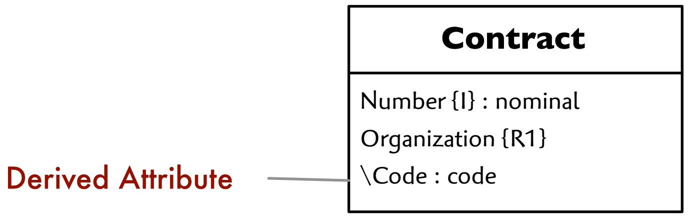
A Derived Attribute is preceded by the ‘\’ character on a class diagram.

Here is an example of a mathematically Derived Attribute on a single class:



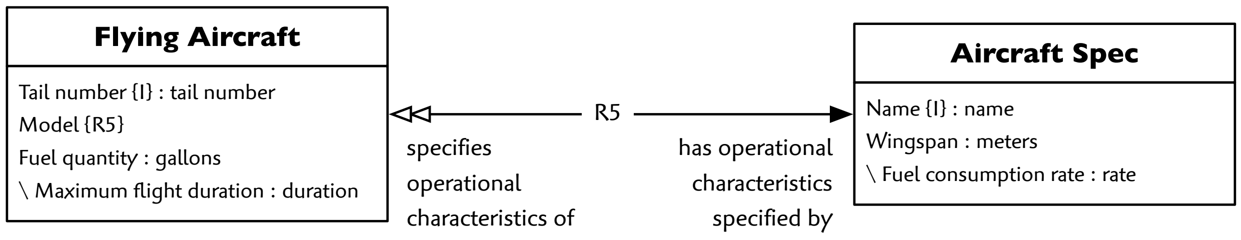
Volume = Height * Radius

Here, an Identifier is derived by concatenating two subfields:



Code = concat(Number, '-', Organization)

Referential Attributes and Identifier Attributes within the same or other Classes may contribute in the construction of a Derived Attribute. In fact, a Derived Attribute may be computed based on other Derived Attributes as shown:


Flying Aircraft
Tail number {I} : tail number
Model {R5}
Fuel quantity : gallons
\ Maximum flight duration : duration
R5
specifies operational characteristics of
has operational characteristics specified by
Aircraft Spec
Name {I} : name
Wingspan : meters
\ Fuel consumption rate : rate

Maximum flight duration =

Fuel quantity / self->Aircraft Spec[R5].Fuel consumption rate

Fuel consumption rate = ... (other performance factors)

In the example above, Aircraft.Flying time is computed based on Fuel Tank.Quantity and Aircraft.Consumption rate. The Consumption rate itself may be computed based on other performance factors.

IMPORTANT: A Derived Attribute may not be, to any degree, derived from itself!

Attributes

Name

Native Attribute.Name

Class

Native Attribute.Class

Domain

Native Attribute.Domain

Formula

The algorithm, concatenation or other processing necessary to compute the current value of the Derived Attribute is defined as the *formula*. For now, this is informal text pending completion of the miUML Action Metamodel subsystem. (Action language will probably be used to specify Formulas in the future). Here are some examples:

```
Volume = Height * Pi * Radius^2

Code = <Organization>-<Number>

Flying time = self->fueled by->Fuel Tank.Quantity / Consumption rate
```

The text symbols employed in the above examples are entirely informal.

When is a Formula executed? In the example above Flying time could be computed upon each read access or, instead, each time the fuel quantity is updated. The choice is entirely implementation specific. Platform performance features must be taken into account to make the right choice. From an analysis perspective, the only thing that matters is that a correct value is obtained whenever \Flying_time is accessed. Consequently, there is no need to express the time of execution for a Formula. It suffices to simply associate it with a Derived Attribute.

IMPORTANT: Each Attribute contributing in a Formula must be instantiated as an Attribute_in_Derivation for the Derived Attribute. In other words, no Attribute that is not an Attribute_in_Derivation where the Derived Attribute is the target may be used in the Derived Attribute's Formula.

Type: Description

Identifiers

1> Name + Class + Domain

Same as the superclass.

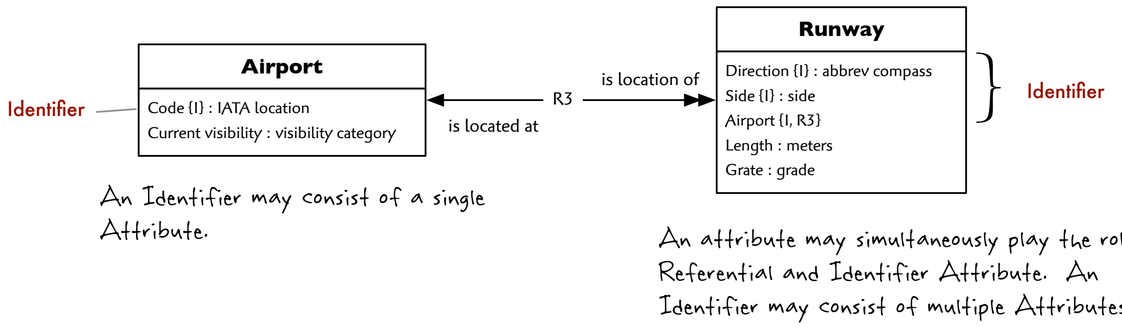
Relationships

None.

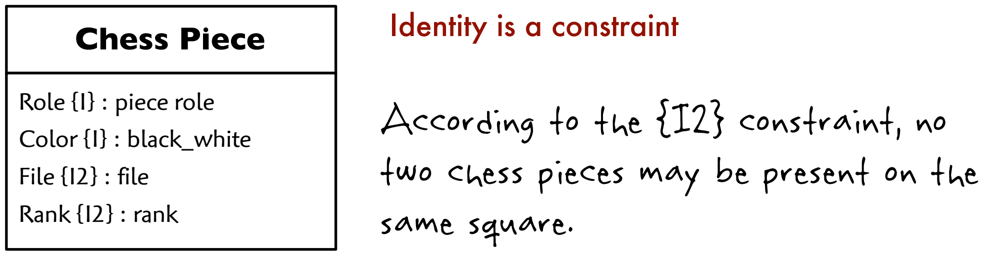
Identifier ID

"An *Identifier* is a set of one or more Attributes that uniquely distinguishes each instance of a Class." [MB]

miUML uses the definition in Executable UML above.


Identifier
Airport
Code {I} : IATA location
Current visibility : visibility category
R3
is location of
is located at
Runway
Direction {I} : abbrev compass
Side {I} : side
Airport {I, R3}
Length : meters
Grate : grade
Identifier
An Identifier may consist of a single Attribute.
An attribute may simultaneously play the role of Referential and Identifier Attribute. An Identifier may consist of multiple Attributes.

An Identifier is not simply a means of locating an instance. Identity is a constraint that ensures that no two instances share certain values. To express a rule that two chess pieces may not be placed on the same square, for example, tag the attribute pair (rank, file) as an Identifier.



Attributes

Number

A Class's Identifiers are numbered sequentially starting from one. By convention, a preferred Identifier is numbered first as {I} (one omitted on the class diagram). Subsequent Identifiers are {I2}, {I3} and so on. Identifier number one is, by default, but not necessarily, the target of any References. In fact, each Identifier of a class is just as good as any other, from an analysis perspective and there is nothing special about a preferred identifier.

Type: Nominal

Class

Class.Name — The Identifier establishes uniqueness within this Class.

Domain

Class.Domain

Identifiers

I> Number + Class + Domain

The Number establishes uniqueness among Identifiers in the same Class.

Relationships

R27

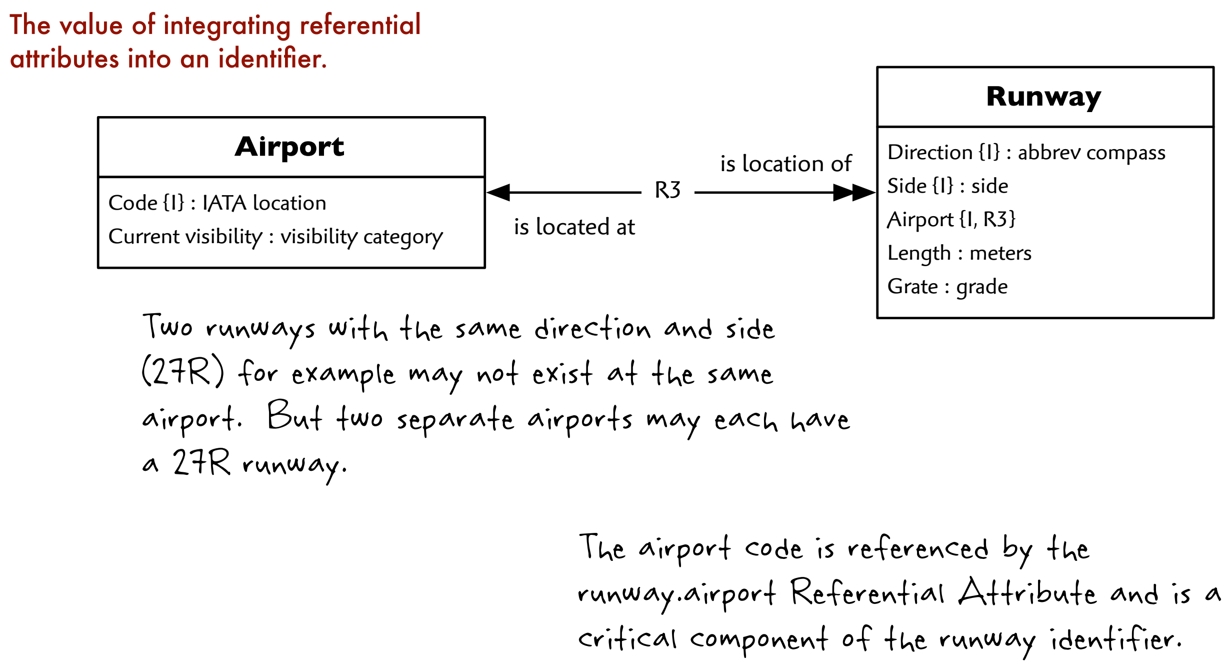
Identifier UNIQUELY DISTINGUISHES EACH INSTANCE OF *exactly one* Class
Class HAS INSTANCES UNIQUELY DISTINGUISHED BY *one or many* Identifier

Here miUML varies a bit from Executable UML:

"Shlaer-Mellor, ... a precursor to Executable UML, required that every class contain at least one identifier, even if that identifier was an attribute placed in the object solely for the purpose of being its identifier. This practice is not required in Executable UML." [MB]

Executable UML as defined in [MB] above offers the option to drop referential attributes and, presumably, express referential constraints in OCL (Object Constraint Language). miUML, on the other hand, encourages the use of referential attributes to enforce model constraints.

Consider the following example from [HTBCM]:


The value of integrating referential attributes into an identifier.
Airport
Code {I} : IATA location
Current visibility : visibility category
R3
is location of
is located at
Runway
Direction {I} : abbrev compass
Side {I} : side
Airport {I, R3}
Length : meters
Grate : grade
Two runways with the same direction and side (27R) for example may not exist at the same airport. But two separate airports may each have a 27R runway.
The airport code is referenced by the runway.airport Referential Attribute and is a critical component of the runway identifier.

In the model example above, the airport code must be included as part of the runway Identifier so that one runway may be distinguished from another across the set of all airports. Without the Reference to the airport Identifier it would be necessary to write OCL (object constraint language) to express the constraint.

The use of referential attributes and identifiers to establish model constraints has a few advantages over OCL:

- Expression directly in the data structure — It is more difficult to violate a constraint if the data structure won't accept incorrect data in the first place. OCL must be interpreted and translated correctly.
- Concise expression — In many cases, a simple referential attribute placement or merge replaces a few lines of OCL.
- Easy to interpret — If you understand a few relational model rules, the implications of referential attribute composition and placement are easy to verify. Just draw tables! The underlying model of OCL is significantly more complex (if one exists at all).
- Consistency with the relational model — and hence many years of well established math, set, logic and data theory.
- Simplicity of a single language — One modeling language (class diagram) can be used to express many fundamental constraints without the need of an additional constraint language.

R30

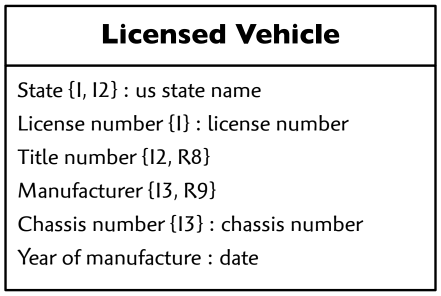
Identifier IS A Modeled OR A Required Referential Identifier

An Identifier is either systematically created from one or more References based on a Relationship's structure (Required Referential) or it is constructed and designated by the modeler to enforce whatever constraints are relevant to the application subject matter.

A Modeled Identifier may incorporate Referential Attributes while a Required Referential Identifier is built exclusively from Referential Attributes and typically whole References. A Required Referential Identifier may not be altered to suit the application subject matter. The rules for constructing Required Referential Identifiers are well established in relational theory and must always be respected to keep the Classes adequately normalized.

Identifier Attribute IDATTR

An Identifier may consist of multiple Attributes. Each Attribute component of an Identifier constitutes an Identifier Attribute, alternatively called an 'attribute in ID' or an 'identifier component'.


Licensed Vehicle
State {I, I2} : us state name
License number {I} : license number
Title number {I2, R8}
Manufacturer {I3, R9}
Chassis number {I3} : chassis number
Year of manufacture : date

Identifier Attributes

1> State
2> State
1> License number
2 Title number
3> Manufacturer
3> Chassis number

The state Attribute participates in Identifiers 1 and 2 yielding two distinct Identifier Attributes.

If an Identifier consists of only a single Attribute, then it has only one Identifier Attribute which constitutes the complete Identifier. Such Identifiers are informally called 'single attribute identifiers'.

Attributes

Identifier

Identifier.Number

Attribute

Attribute.Name

Class

Identifier.Class and Attribute.Class — The Identifier is in the same Class as the component Attribute.

Domain

Identifier.Domain and Attribute.Domain

Identifiers

I> Identifier + Attribute + Class + Domain

Standard Mx:Mx composition of an association class identifier with the Class and Domain attributes individually merged.

Relationships

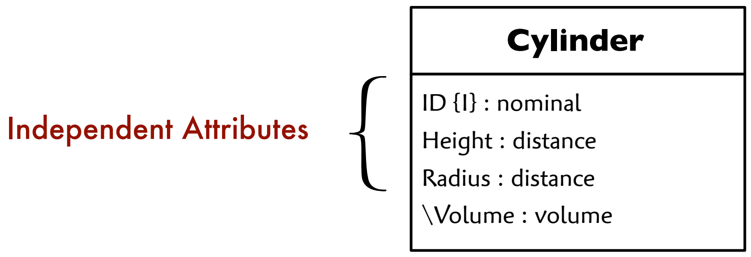
R22
Attribute IS REQUIRED IN *zero, one or many* Identifier
Identifier REQUIRES *one or many* Attribute

By definition, an Identifier consists of at least one Attribute.

An Attribute may or may not be part of an Identifier. And it is possible for an Attribute to be a component simultaneously of multiple Identifiers. (There are numerous examples of this in the miUML metamodel itself).

Independent Attribute IND_ATTR

An Attribute is *independent* when its value is not derivable from the values of other Attributes in the same Domain at a given slice of time. An Independent Attribute may, however, be updated based on changing circumstances such as state changes, class collaborations and domain interactions, such as a command from a user interface domain, for example.



Attributes

Name

Native Attribute.Name

Class

Native Attribute.Class

Domain

Native Attribute.Domain

Identifiers

I> Name + Class + Domain

Same as the superclass.

Relationships

None.

Modeled Identifier MOD_ID

A *Modeled Identifier* is crafted by the modeler to express a constraint relevant to the application subject matter. It may incorporate any combination of Independent, Derived and Referential Attributes.

Attributes

Number

Identifier.Number

Class

Identifier.Class

Domain

Identifier.Domain

Identifiers

I> Number + Class + Domain

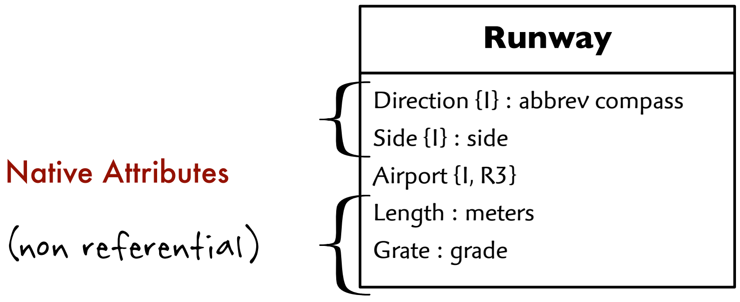
Same as the superclass.

Relationships

None.

Native Attribute NAT_ATTR

An Attribute is *native* in the sense that it is an intrinsic property defined locally for a Class without reference to any other Attribute. In other words, a Native Attribute is not a Referential Attribute. On a class diagram, a Native Attribute is shown with a the name of some Type (data). A Native Attribute cannot have an {R} tag.



Attributes

Name

Attribute.Name

Class

Attribute.Class

Domain

Attribute.Domain

Type - constrained

Type.Name — This Attribute's data Type. This can be any System Type or a Domain Type defined in this Attribute's Domain.

Identifiers

I> Name + Class + Domain

Same as the superclass.

Relationships

R24

Native Attribute MAY ASSUME VALUES DEFINED BY *exactly one* Type
Type DEFINES RANGE OF VALUES ASSIGNABLE TO *zero, one or many* Native Attribute

Every value is defined by some Type (data type) [DATE1]. The range of values that may be assigned to any instance of a Attribute are defined by a single Type. There is no such thing as a type-less value, and every Attribute can take on values, so every Attribute requires a Type.

A Type is defined directly only for each Native Attribute. The Type of a Referential Attribute is then determined by following a chain of one or more Referential Attributes ultimately leading to a Native Attribute.

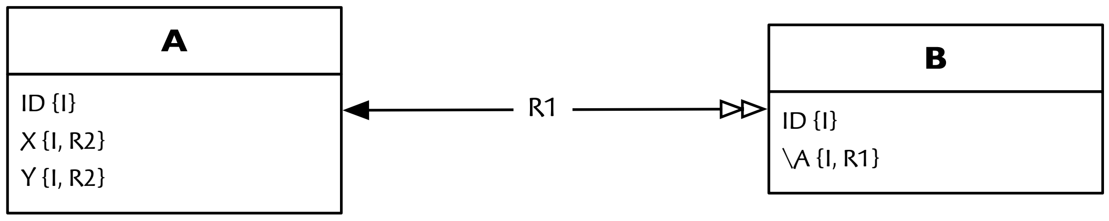
A Type may be defined that is not currently used by any Native Attribute. This makes it possible to predefine a library of Types for later use. A Type that does not constrain any Native Attribute, however, will have no relevance in a populated metamodel. (A code generator would, therefore, safely disregard any such unused Types).

R25

Native Attribute IS AN Independent OR A Derived Attribute

The value of a Native Attribute is either independent or computed based the values of one or more other Attributes in the same Domain (Derived). Referential Attributes, by contrast, are never derived. This makes sense, since a Referential Attribute always refers to one or more other Attributes.

That said, one could imagine using the derivation notation to condense multiple Referential Attributes within a Class into a single shorthand attribute. Consider two Classes:



Derivation formula:
B.A = { A.ID, A.X, A.Y }

Where { , } yields a set of attribute values

It is easy enough to specify the shorthand notation, but it would then be necessary to extend the miUML metamodel to accommodate rules for including a set of Referential Attributes in such a derivation. It is not clear that it is worth the effort to support such a construct. If \A were constructed in this fashion, it would be difficult to specify a constraint using a subset of \A's component Attributes, since they would be hidden.

At this point it seems best to leave derivation undefined for Referential Attributes.

Non Specialized Class NSP_CLASS

Any Class that does not participate as a Superclass or a Subclass in any Generalization Relationship is a *non-specialized class*. An instance of a Non Specialized Class represents a complete object. This type of Class is abstracted to recognize this simplified instantiation rule.

Attributes

Name

Class.Name

Domain

Class.Domain

Identifiers

I> Name + Domain

Same as the superclass.

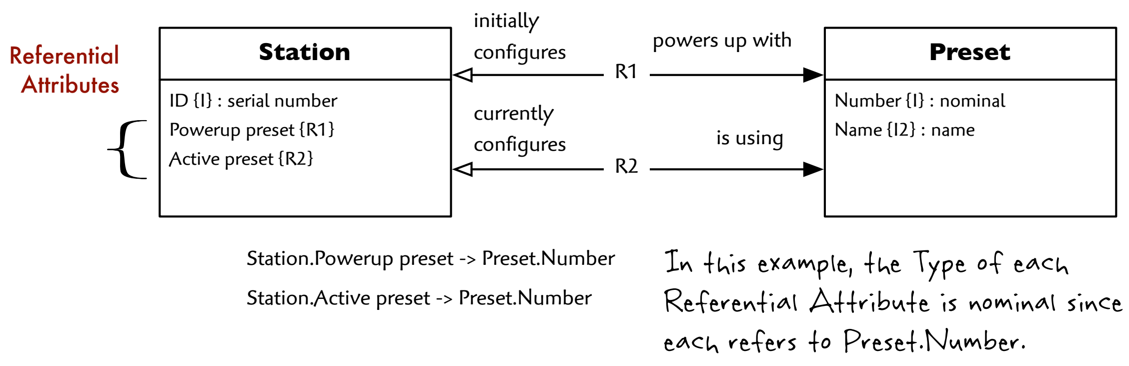
Relationships

None.

Referential Attribute REF_ATTR

"A *referential attribute* identifies an Attribute whose value is the value of an identifying Attribute in one or more associated Classes." [MB]

miUML uses the definition in Executable UML above.



A Referential Attribute takes on the Type of some other Attribute via a Relationship formalization. A Referential Attribute is associated with an {R} tag and no Type on a miUML class diagram. A single Referential Attribute may participate in multiple Relationships and References.

ATTRIBUTES

Name

Attribute.Name

Class

Attribute.Class

Domain

Attribute.Domain

\Type

The Type of a Referential Attribute is the Type of the referenced Identifier Attribute. If the Identifier Attribute itself is referential, then the same rule applies for that Attribute. This suggests, as one option, a recursive traversal terminated by the necessary discovery of a native Identifier Attribute. Traversal from a Referential Attribute through one or more References must ultimately land on a Native Attribute. (If none is found, then there is an unresolved cycle in the M1 layer class model which must be fixed).

Identifiers

1> Name + Class + Domain

Same as the superclass.

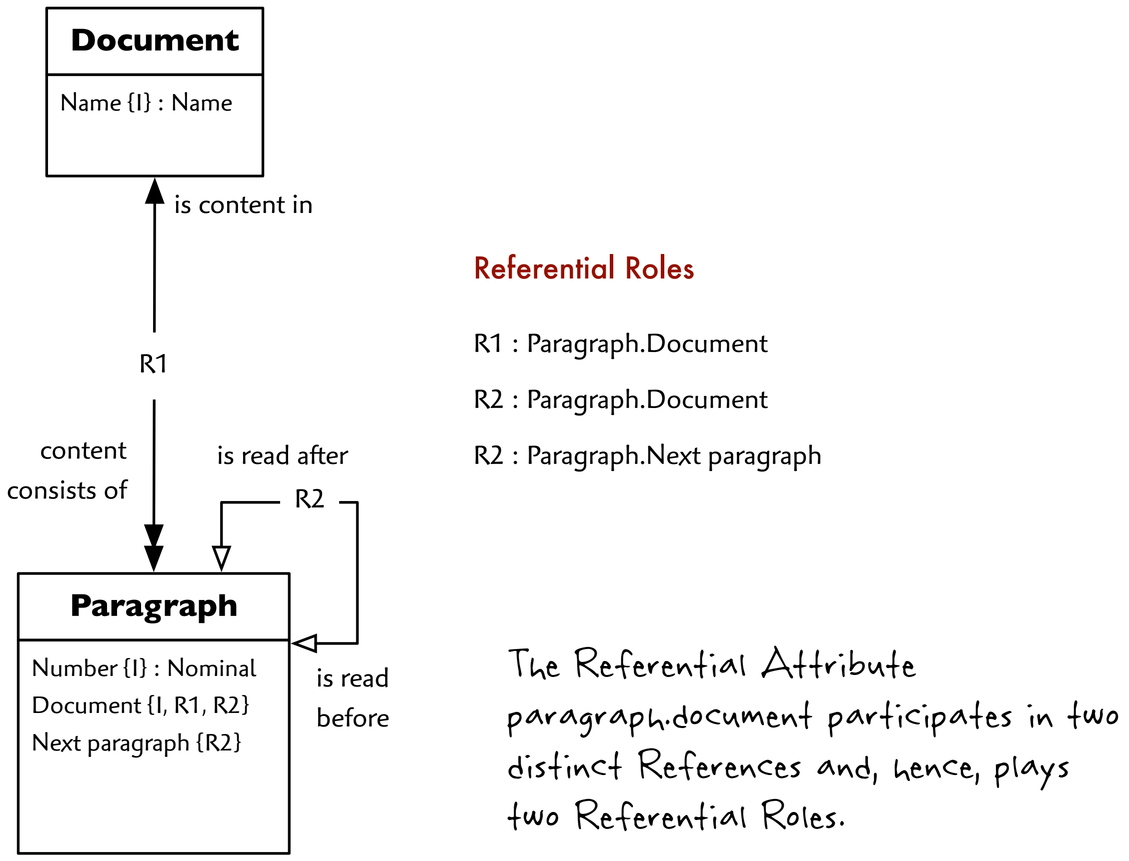
Relationships

None.

Referential Role REFROLE

A Referential Attribute may simultaneously participate in more than one Reference. In the example below, the Referential Attribute Paragraph.Document simultaneously refers to the document in which it is contained, R1, and the document that contains the next paragraph, R2. This prevents the illegal specification of a paragraph that precedes one in a separate document.

Each such usage of a Referential Attribute constitutes a Referential Role.



Attributes

From attribute

Referential Attribute.Name — The name of the referring Referential Attribute.

From class

Referential Attribute.Class and Reference.From class — The Referential Attribute is defined in this Class which is also the source of the Reference.

Reference type

Reference.Type

To attribute — constrained

Identifier Attribute.Attribute — This is the name of the Identifier Attribute referred to by the Referential Attribute. The value pair (From attribute, From class) may not match the (To attribute, To class) value pair. In other words, a Referential Attribute may never refer to itself! This is a simple case of an illegal cycle where a Native Attribute, and hence Type, is never found. It's bad mojo.

To class — constrained

Identifier Attribute.Class and Reference.To class — The referenced Identifier Attribute's Class is the Reference's To class. The constraint is described under To attribute, above.

To identifier — constrained

Identifier Attribute.Identifier — This is the number of the referenced Identifier Attribute. Same constraint on attribute and class applies here (all part of the same reference).

Rnum

Reference.Rnum — The Relationship partially or fully formalized by this Referential Role.

Domain

Referential Attribute.Domain and Identifier Attribute.Domain and Reference.Domain — The Domain of the aforementioned Relationship, Classes and Attributes.

IDENTIFIERS

I> From attribute + From class + Reference Type + To class + Rnum + Domain

Standard composition of a Mx:Mx association class identifier. Constructed by taking the Reference identifier (Type + *kkkFrom* class + To class + Rnum + Domain) and the Referential Attribute identifier (Name + Class + Domain) with attributes merged as follows:

Reference.Domain + Referential Attribute.Domain = Referential_Role.Domain

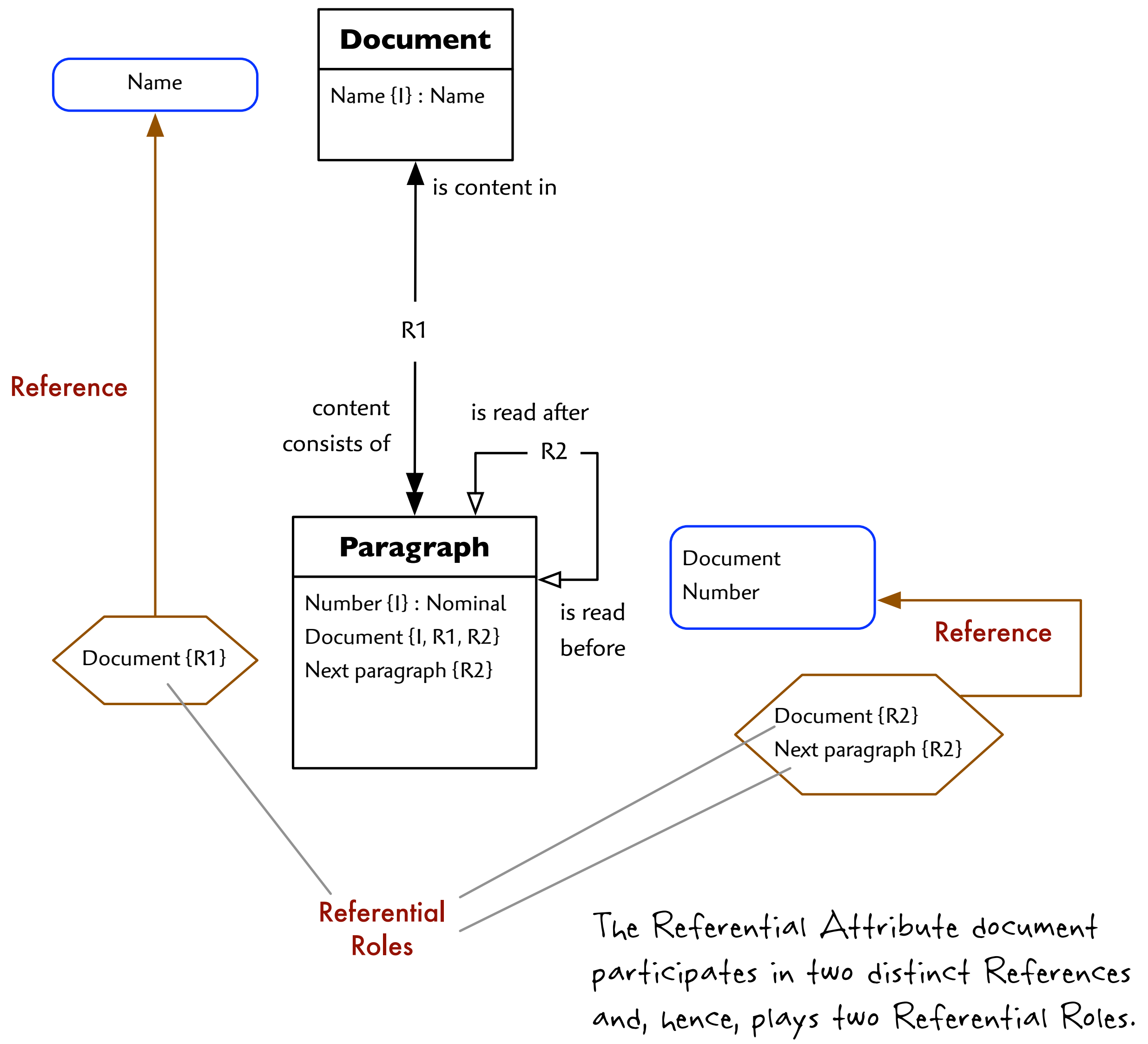
Reference.From class + Referential Attribute.Class = Referential_Role.From_class

RELATIONSHIPS

R31

Reference CONSISTS OF *one or many* Referential Attribute
Referential Attribute IS PART OF *one or many* Reference

Each Reference must point to a complete Identifier. For each component of the target Identifier, there must be a corresponding Referential Attribute in the Reference. Since an Identifier is made up of one or more Attributes, a Reference must also consist of at least one Referential Attribute.

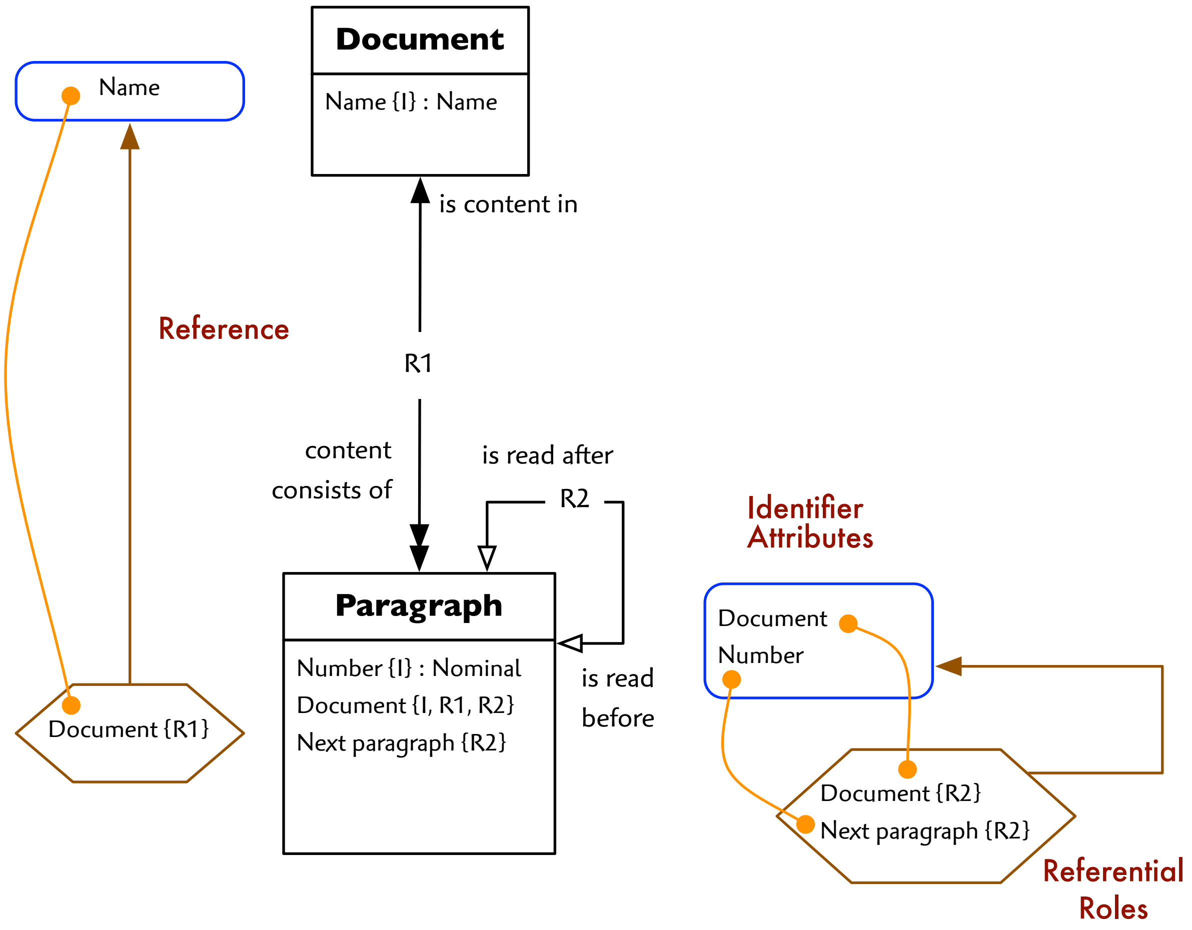


A Referential Attribute may play a role in more than one Relationship. Each such role will apply to a distinct Reference.

R32

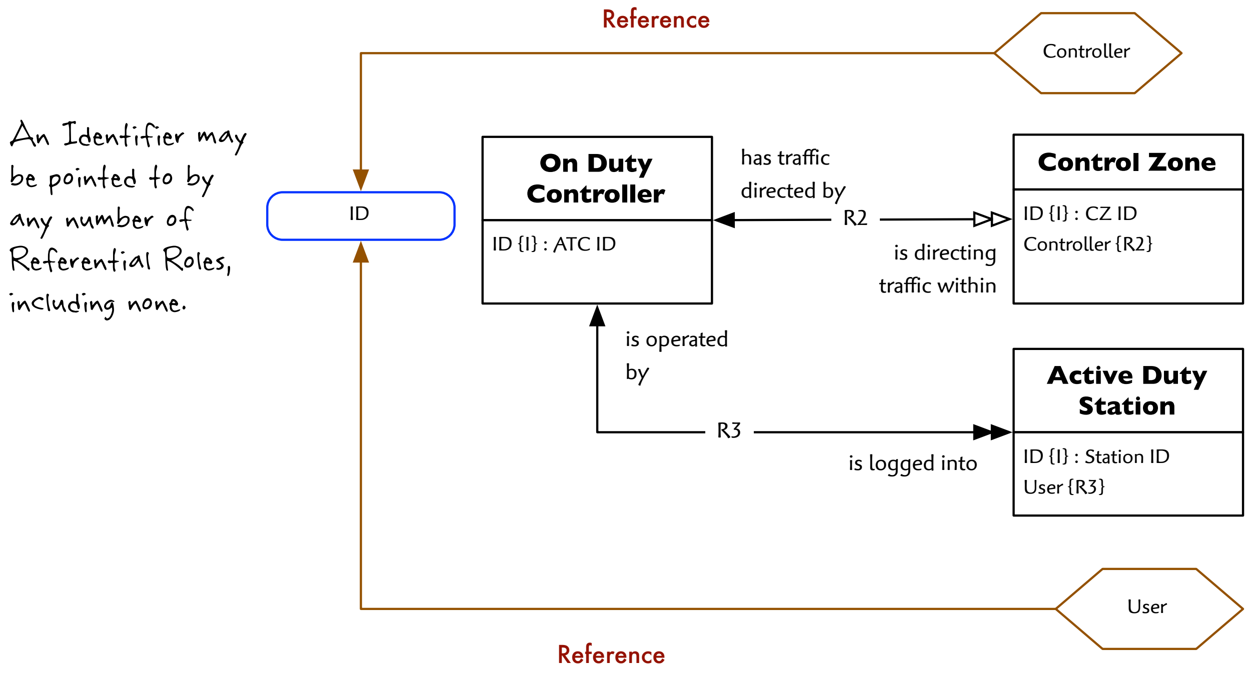
Referential Role REFERENCES *exactly one* Identifier Attribute
Identifier Attribute IS REFERENCED BY *zero, one or many* Referential Role

By definition, a Referential Role represents the direction of reference from a Referential Attribute to a specific Identifier Attribute. A complete Reference binds one or more Referential Attributes to a complete Identifier with all of its component Identifier Attributes.



Each Referential Role points to a single Identifier Attribute.

A Class may or may not serve as a reference target even though it may participate in one or more Relationships. If a Class does not participate in any Relationships, then none of its Identifier Attributes will be referenced. If a Class participates in some Relationships, but only as the source of a Reference in each of these Relationships, then again, none of the Class's Identifier Attributes will be referenced.



Referential loops are forbidden, so an Identifier Attribute that also happens to be a Referential Attribute may not refer to itself.