ADVANCED RELATIONSHIPS

Relationship

A relationship is a connection among things(things can be any of, structural, behavioural, annotational or grouping. This module contains mainly structural things). In object-oriented modeling, the four most important relationships are dependencies, generalizations, associations, and realizations. Diagram indicating advanced relationship is shown in Figure: 1

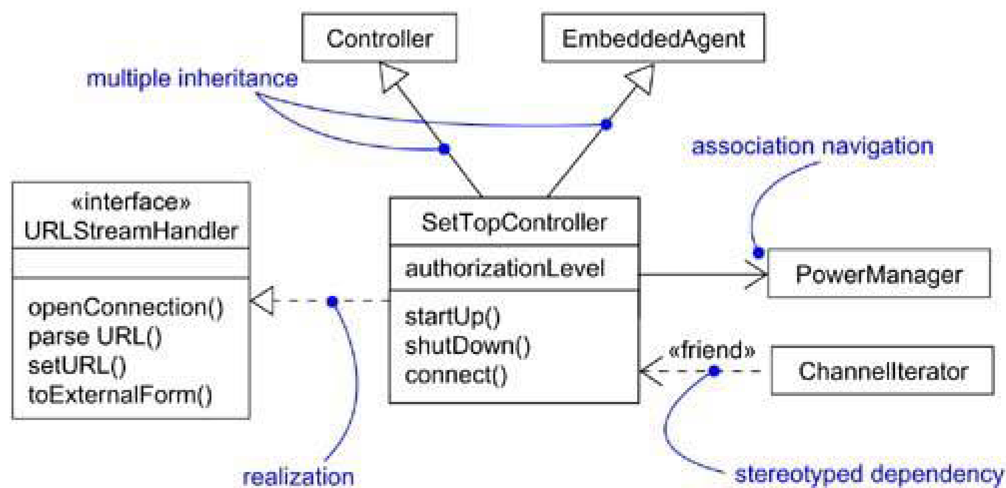


Figure: 1 Advanced Relationship

Dependency

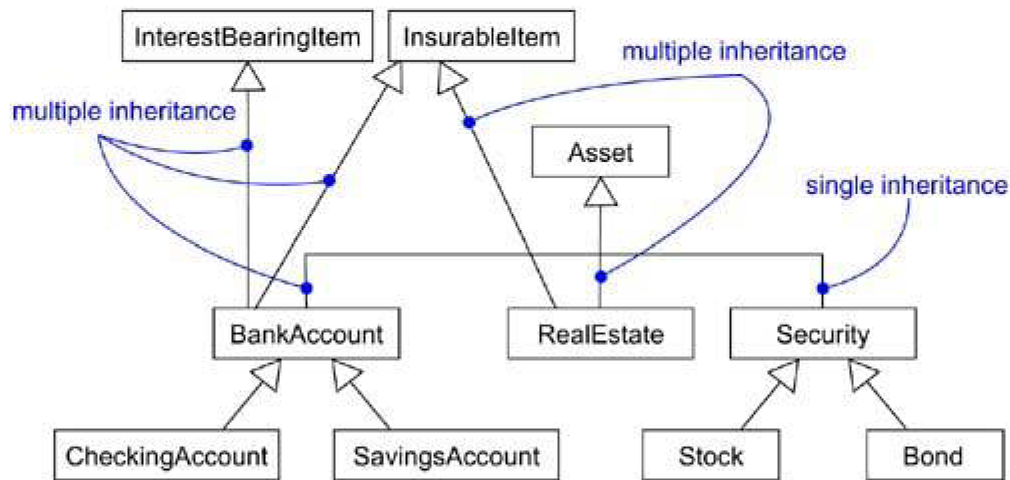
A dependency is a using relationship, specifying that a change in the specification of one thing may affect another thing that uses it.

Stereotypes for Dependency relationship (17 stereotypes in 6 groups)
dependency relationships ***among classes and objects*** in class diagrams
«bind» – Specifies that the source instantiates the target template using the given actual parameters
«derive» – Specifies that the source may be computed from the target
«friend» – Specifies that the source is given special visibility into the target
«instanceOf» – Specifies that the source object is an instance of the target classifier

«instantiate» – Specifies that the source creates instances of the target
«powertype» – Specifies that the target is a powertype of the source; a powertype is a classifier whose objects are all the children of a given parent
«refine» – Specifies that the source is at a finer degree of abstraction than the target
«use» – Specifies that the semantics of the source element depends on the semantics of the public part of the target
dependency relationships ***among packages***
«access» – Specifies that the source package is granted the right to reference the elements of the target package
«import» – A kind of access that specifies that the public contents of the target package enter the flat namespace of the source, as if they had been declared in the source
dependency relationships ***among use cases***
«extend» – Specifies that the target use case extends the behavior of the source
«include» – Specifies that the source use case explicitly incorporates the behavior of another use case at a location specified by the source
when modeling interactions ***among objects***
«become» – Specifies that the target is the same object as the source but at a later point in time and with possibly different values, state, or roles
«call» – Specifies that the source operation invokes the target operation
«copy» – Specifies that the target object is an exact, but independent, copy of the source
in the ***context of state machines***
«send» – Specifies that the source operation sends the target event
in the ***context of organizing the elements of your system into subsystems and models*** is
«trace» – Specifies that the target is an historical ancestor of the source.
Generalization

A generalization is a relationship between a general thing and a more specific kind of that thing. Here child will inherit all the structure and behaviour of the parent and can even add new structure and behavior, or it may modify the behavior of the parent(i.e,

overriding). Figure: 2 shows various inheritance possible in UML.



UML defines ***one stereotype and four constraints*** that may be applied to generalization relationships.

«implementation» – Specifies that the child inherits the implementation of the parent but does not make public nor support its interfaces, thereby violating substitutability.

Note: use implementation when you want to model private inheritance

{complete} – Specifies that all children in the generalization have been specified in the model and that no additional children are permitted

Note: use the complete constraint when you want to show explicitly that you've fully specified a hierarchy in the model

{incomplete} – Specifies that not all children in the generalization have been specified and that additional children are permitted

Note: use incomplete to show explicitly that you have not stated the full specification of the hierarchy in the model

{disjoint} – Specifies that objects of the parent may have no more than one of the children as a type

{overlapping} – Specifies that objects of the parent may have more than one of the children

as a type

Note: use disjoint and overlapping when you want to distinguish between static classification (disjoint) and dynamic classification (overlapping).

Association

An association is a structural relationship, specifying that objects of one thing are connected to objects of another.

Navigation : Unless otherwise specified, navigation across an association is bidirectional. Unidirectional navigation is also possible where reverse navigation is not desirable such as where security concern is an important thing. A Unidirectional navigation is shown in Figure:3.

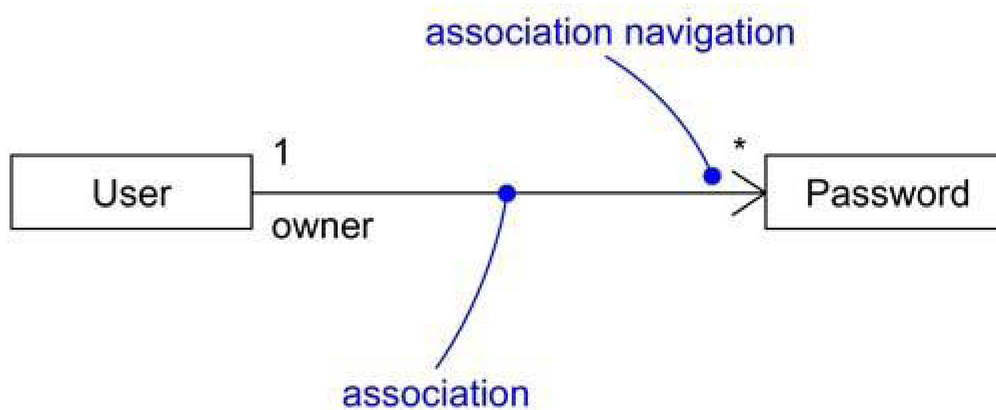


Figure:3 unidirectional-navigation

Visibility : For an association between two classes, objects of one class can see and navigate to objects of the other, unless otherwise restricted by an explicit statement of navigation. ***Three levels of visibility*** for an association possible in UML.
Default visibility of a role is public. ***Private visibility***indicates that objects at that end are not accessible to any objects outside the association; ***protected visibility***indicates that objects at that end are not accessible to any objects outside the association, except for children of the other end.

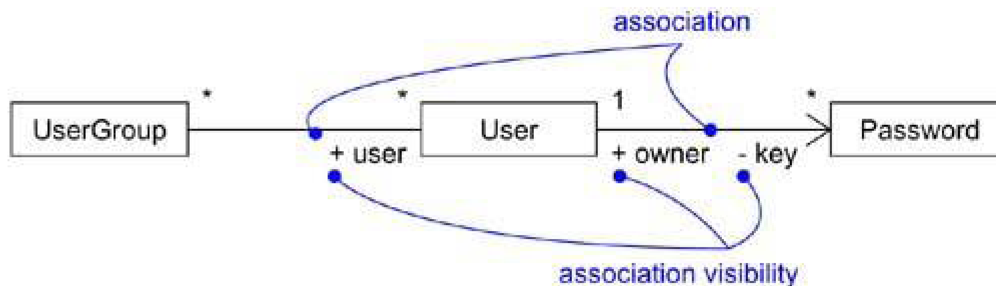


Figure:4 Association_Visibility

Qualification :A qualifier is an association attribute whose values partition the set of objects related to an object across an association. It is graphically represented as a

small rectangle attached to the end of an association, placing the attributes in the rectangle. For example If you can devise a lookup data structure at one end of an association (for example, a hash table or b-tree), then manifest that index as a qualifier. In most cases, the source end's multiplicity will be many and the target end's multiplicity will be 0..1.

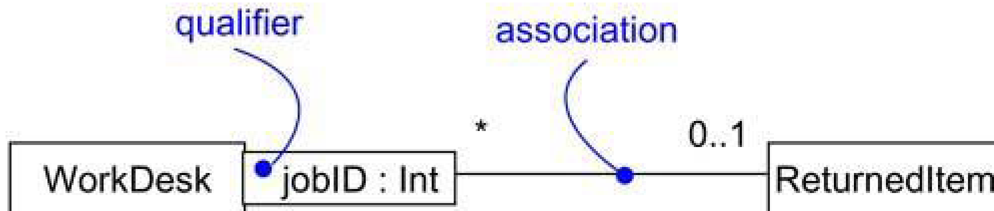


Figure:5 Qualification

Interface Specifier : An interface is a collection of operations that are used to specify a service of a class or a component; every class may realize many interfaces. In the context of an association with another target class, a source class may choose to present only part of its face(interfaces) to the world and this part that is chosen is called as the interface specifier. Figure: 6 illustrates this.

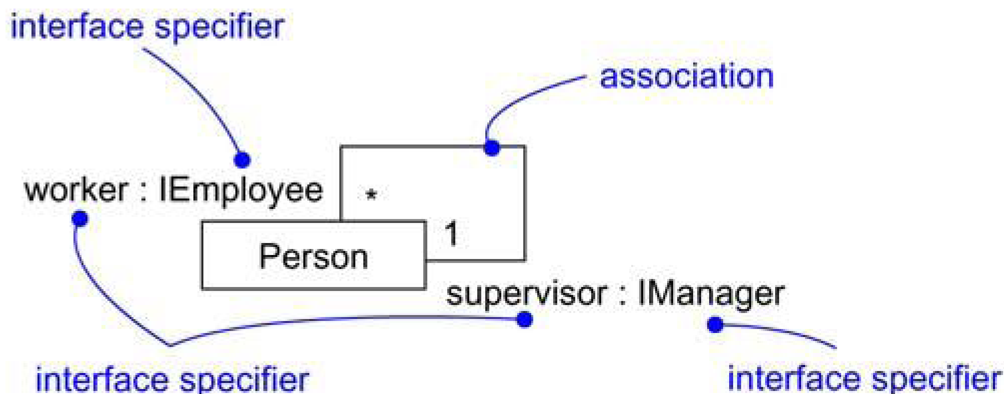


Figure:6 Interface Specifier

Here in Figure:6, In the vocabulary of a HR system, a Person class may realize many interfaces such as IManager, IEmployee, IOfficer, and so on. A relationship between a supervisor and workers with a one-to-many association is shown by explicitly labeling the roles of this association as supervisor and worker. In the context of this association, <u>a Person in the role of supervisor presents only the IManager face to the worker; a Person in the role of worker presents only the IEmployee face to the</u>

supervisor.We can *explicitly show the type of role using the syntax* rolename : iname, where iname is some interface of the other classifier.

Composition : In a composite aggregation, an object may be a part of only one composite at a

time. For example, in a windowing system, a Frame belongs to exactly one Window. In a composite aggregation, the whole is responsible for the disposition of its parts, which means that the composite must manage the creation and destruction of its parts.

For example, when you create a Frame in a windowing system, you must attach it to an enclosing Window. Similarly, when you destroy the Window, the Window object must in turn destroy its Frame parts.

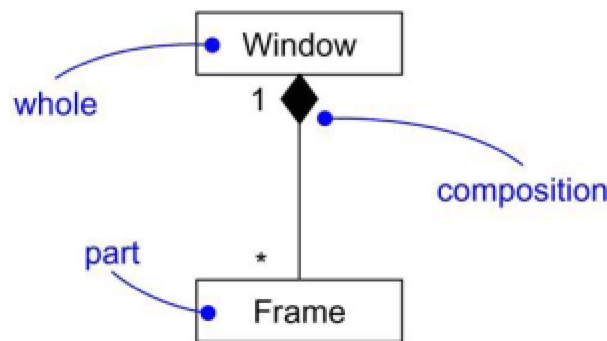


Figure:7 Composition

Association Classes : An association class can be seen as an association that also has class properties, or as a class that also has association properties. An association class is represented in Figure:8.

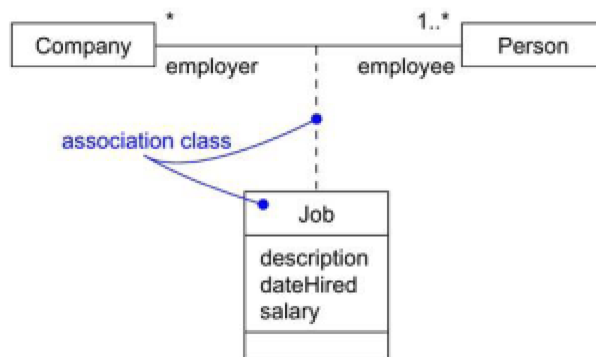


Figure:8 Association class

Here in figure:8 the associations on both ends can be a many to many that indicates a person can be an employee in more than one company at a time. So in this case it is

not possible to place the job details in any of the class(Company or Person). In such a situation we go for association class.

five constraints that can be applied to association relationships.

{implicit} – Specifies that the relationship is not manifest but, rather,is only conceptual

{ordered} – Specifies that the set of objects at one end of an association are in an explicit order

constraints that ***relate to the changeability of the instances*** of an association.

{changeable} – Links between objects may be added, removed, and changed freely

{addOnly} – New links may be added from an object on the opposite end of the association

{frozen} – A link, once added from an object on the opposite end of the association, may not be modified or deleted

constraint ***for managing related sets*** of associations

{xor} – Specifies that, over a set of associations, exactly one is manfest for each associated object.

Realization

A realizationis a semantic relationship between classifiers in which one classifier specifies a contract that another classifier guarantees to carry out. Realization can be represented in two ways: in the canonical form (using the interface stereotype and the dashed directed line with a large open arrowhead) and in an elided form (using the interface lollipop notation). It is represented in Figure:9.

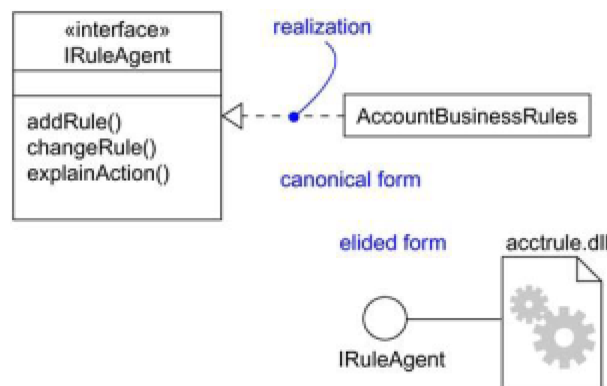


Figure:9 Realization

Source : http://praveenthomasln.wordpress.com/2012/03/03/advanced-relationships-s8-csediting-in-progress/