¿Cuál es la diferencia entre asociación, agregación y composición?

¿Cuál es la diferencia entre asociación, agregación y composición? Por favor explique en términos de implementación.

- **La asociación** es una relación donde todos los objetos tienen su propio ciclo de vida y no hay propietario.

  Tomemos un ejemplo de Maestro y Estudiante. Varios estudiantes pueden asociarse con un solo profesor y un solo alumno puede asociarse con varios profesores, pero no hay propiedad entre los objetos y ambos tienen su propio ciclo de vida. Ambos se pueden crear y eliminar de forma independiente.

- **La agregación** es una forma especializada de asociación donde todos los objetos tienen su propio ciclo de vida, pero hay propiedad y los objetos secundarios no pertenecerán a otro objeto principal mientras se referencian.

  Tomemos un ejemplo de Departamento y maestro. Un solo maestro no puede pertenecer a múltiples departamentos, pero si eliminamos el departamento, el objeto maestro *no* se destruirá. Podemos pensarlo como una relación "tiene una".

- **La composición** es nuevamente una forma especializada de Agregación y podemos llamar a esto como una relación de "muerte". Es un tipo fuerte de Agregación. El objeto secundario no tiene su ciclo de vida y si se elimina el objeto principal, también se eliminarán todos los objetos secundarios.

  Tomemos nuevamente un ejemplo de relación entre Casa y Habitaciones. La casa puede contener varias habitaciones; no hay vida independiente de la habitación y ninguna habitación puede pertenecer a dos casas diferentes. Si eliminamos la casa, la sala se eliminará automáticamente.

  Tomemos otra relación de ejemplo entre Preguntas y Opciones. Las preguntas individuales pueden tener múltiples opciones y la opción no puede pertenecer a múltiples preguntas. Si eliminamos las preguntas, las opciones se eliminarán automáticamente.

Para dos objetos, `Foo` y `Bar` las relaciones se pueden definir

Asociación – Tengo una relación con un objeto. `Foo` usa `Bar`

```
public class Foo { void Baz(Bar bar) { } };
```

Composición: poseo un objeto y soy responsable de su vida, cuando `Foo` muere, también lo hace `Bar`

```
public class Foo { private Bar bar = new Bar(); }
```

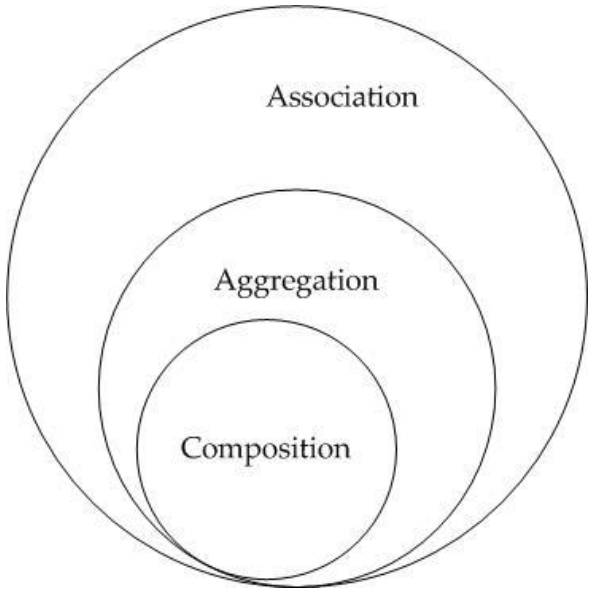
Agregación: tengo un objeto que he pedido prestado a otra persona. Cuando `Foo` muere, `Bar` puede `Bar` vivo.

```
public class Foo { private Bar bar; Foo(Bar bar) { this.bar = bar; } }
```

Sé que esta pregunta está etiquetada como C # pero los conceptos son preguntas bastante genéricas como esta redirigir aquí. Así que voy a proporcionar mi punto de vista aquí (un poco sesgado desde el punto de vista de Java donde estoy más cómodo).

Cuando pensamos en la naturaleza orientada a Objetos siempre pensamos en Objetos, clase (planos de objetos) y la relación entre ellos. Los objetos están relacionados e interactúan entre sí a través de métodos. En otras palabras, el objeto de una clase puede usar servicios / métodos proporcionados por el objeto de otra clase. Este tipo de relación se denomina **asociación.**

La agregación y la composición son subconjuntos de asociación, lo que significa que son casos específicos de asociación.



- Tanto en agregación como en composición, el **objeto de una clase "posee" el objeto de otra clase**.
- Pero hay una diferencia sutil. En **Composición,** el objeto de clase que pertenece al objeto de su clase propietaria **no puede vivir por sí mismo** (también llamado "relación de muerte"). Siempre vivirá como parte de su objeto propietario donde, como en **Agregación,** el objeto dependiente es **independiente** y puede existir incluso si el objeto de la clase propietaria está muerto.
- Entonces en la composición si el objeto propietario es basura recogida, el objeto propiedad también lo será, que no es el caso en la agregación.

¿Confuso?

Ejemplo de composición: Considere el ejemplo de un automóvil y un motor que es muy específico para ese automóvil (lo que significa que no se puede usar en ningún otro automóvil). Este tipo de relación entre **Auto** y clase **Motor,** se llama Composición. El objeto de la clase **Auto** no puede existir sin el objeto de la clase **Motor**, y el objeto de **Motor** no tiene significado sin la clase **Auto**. Para poner en palabras simples, la clase de **Auto** únicamente "posee" la clase **Motor**.

Ejemplo de agregación: Considere ahora la clase **Coche** y clase **Rueda**. El auto necesita un objeto **Rueda** para funcionar. Es decir, objeto del objeto propio de la rueda del objeto, pero no podemos decir que el objeto rueda no tenga significado sin el objeto del carro. Puede ser utilizado en una bicicleta, camión u objeto de automóvil diferente.

Resumiendo –

En resumen, la asociación es un término muy genérico que se usa para representar cuando en clase se usan las funcionalidades proporcionadas por otra clase. Decimos que es composición si un objeto de clase padre posee otro objeto de clase hijo y ese objeto de clase hijo no puede existir de manera significativa sin el objeto de clase padre. Si puede, entonces se llama Agregación.

Más detalles aquí.

La asociación es un concepto generalizado de relaciones. Incluye tanto Composición como Agregación.

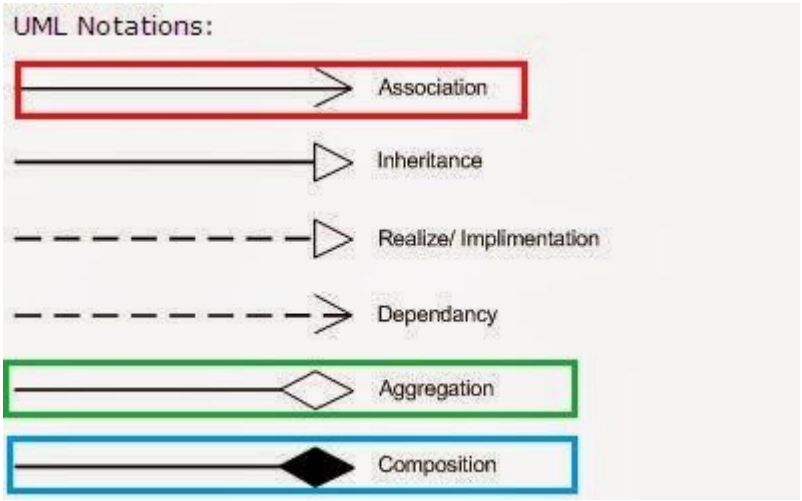
La composición (mezcla) es una forma de envolver objetos simples o tipos de datos en una **sola unidad**. Las composiciones son un componente fundamental de muchas estructuras de datos básicas

La agregación (colección) difiere de la composición ordinaria en que no implica propiedad. En la composición, cuando se destruye el objeto propietario, también lo son los objetos contenidos. En agregación, esto no es necesariamente cierto.

Ambos denotan relación entre objeto y solo difieren en su fuerza.

	Aggregation	**Composition**
Life time	Have their own lifetime	Owner's life time
Child object	Child objects belong to a single parent	Child objects belong to a single parent
Relation	Has-A	Owns
Example	**Car and Driver**	**Car and Wheels**

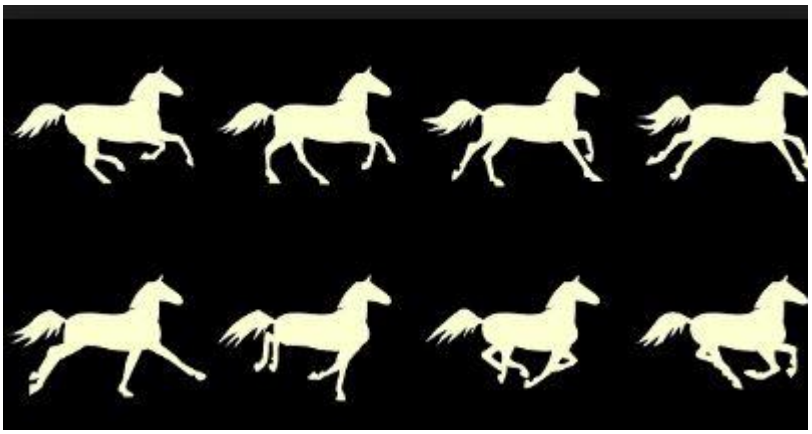
Ahora observe la siguiente imagen



Analogía:

Composición : la siguiente imagen es composición de imágenes, es decir, utilizando imágenes individuales para hacer una imagen.

Agregación: colección de imágenes en una sola ubicación



Por ejemplo, una universidad posee varios departamentos, y cada departamento tiene varios profesores. Si la universidad cierra, los departamentos ya no existirán, pero los profesores en esos departamentos continuarán existiendo. Por lo tanto, una universidad puede verse como una composición de departamentos, mientras que los departamentos tienen una agregación de profesores. Además, un profesor podría trabajar en más de un departamento, pero un departamento no podría formar parte de más de una universidad.

La asociación representa la capacidad de una instancia para enviar un mensaje a otra instancia. Esto se implementa típicamente con un puntero o una variable de

instancia de referencia, aunque también se puede implementar como un argumento de método o la creación de una variable local.

```
//[Example:] //|A|----------->|B| class A { private: B* itsB; };
```

La agregación [...] es la relación típica de todo / parte. Esto es exactamente lo mismo que una asociación con la excepción de que las instancias no pueden tener relaciones de agregación cíclica (es decir, una parte no puede contener su totalidad).

```
//[Example:] //|Node|<>-------->|Node| class Node { private: vector itsNodes; };
```

El hecho de que esto sea una agregación significa que las instancias de Node no pueden formar un ciclo. Por lo tanto, este es un Árbol de Nodos, no un gráfico de Nodos.

La composición es [...] exactamente como Agregación, excepto que la vida de la "parte" está controlada por el "todo". Este control puede ser directo o transitivo. Es decir, el "todo" puede asumir la responsabilidad directa de crear o destruir la "parte", o puede aceptar una parte ya creada, y luego pasarla a otro todo que asume la responsabilidad por ello.

```
//[Example:] //|Car|<#>-------->|Carburetor| class Car { public: virtual ~Car()
{delete itsCarb;} private: Carburetor* itsCarb };
```

- **Dependencia** (referencias)
  Significa que no hay un enlace conceptual entre dos objetos. por ejemplo, el objeto **Inscripción** hace referencia a objetos de Estudiante y Curso (como parámetros de método o tipos de devolución)

```
public class Inscripcion { public void inscribir(Estudiante s, Curso c){} }
```

- **Asociación** (has-a)
  Significa que casi siempre hay un enlace entre los objetos (están asociados). El objeto de pedido ***tiene un*** objeto Cliente

```
public class Orden { private Cliente cliente }
```

- **Agregación** (has-a + whole-part)
  Tipo especial de asociación donde hay una relación de parte entera entre dos objetos. sin embargo, podrían vivir el uno sin el otro.

```
public class PlayList{ private Agregar (cancion C){}; }
```

Nota: la parte más difícil es distinguir la agregación de la asociación normal. Honestamente, creo que esto está abierto a diferentes interpretaciones.

Composición (has-a + whole-part + ownership)

Tipo especial de agregación Un `Apartment` se compone de algunas `Room`. Una `Room` no puede existir sin un `Apartment`. cuando se elimina un apartamento, también se eliminan todas las salas asociadas.

```
public class Apartment{ private Room bedroom;
public Apartment() {  bedroom = new Room(); } }
```

Como otros dijeron, una asociación es una relación entre objetos, la agregación y la composición son tipos de asociación.

Desde el punto de vista de la implementación, se obtiene una agregación al tener un miembro de clase **por referencia**. Por ejemplo, si la clase A agrega un objeto de clase B, tendrá algo como esto (en C ++):

```
class A { B & element; // or B * element; };
```

La semántica de la agregación es que cuando se destruye un objeto A, el objeto B que está almacenando todavía existirá.

Al usar la composición, tiene una relación más fuerte, generalmente al guardar el miembro **por valor**: `class A { B element; };`

Aquí, cuando se destruye un objeto A, también se destruirá el objeto B que contiene. La manera más fácil de lograr esto es almacenando el miembro por valor, pero también puede usar algún puntero inteligente o eliminar el miembro en el destructor:

```
class A { std::auto_ptr element; }; class A { B * element; ~A() { delete B; } };
```

El punto importante es que, en una composición, el objeto contenedor **posee** el contenido, mientras que en la agregación lo hace **referencia**.

Es sorprendente cuánta confusión existe sobre la distinción entre los tres conceptos de relación *asociación*, *agregación* y *composición*.

Tenga en cuenta que los términos *agregación* y *composición* se han utilizado en la comunidad C ++, probablemente durante algún tiempo antes de que se hayan definido como casos especiales de *asociación* en los diagramas de clase UML.

El problema principal es el malentendido generalizado y continuo (incluso entre los desarrolladores de software expertos) de que el concepto de composición implica una dependencia del ciclo de vida entre el todo y sus partes, de modo que las partes no pueden existir sin el todo, ignorando el hecho de que también hay casos de

asociaciones de parte-todo con partes no compartibles donde las partes pueden separarse y sobrevivir a la destrucción del todo.

Por lo que puedo ver, esta confusión tiene dos raíces:

1. En la comunidad C ++, el término "agregación" se usó en el sentido de una clase que define un atributo para referenciar objetos de otra clase independiente (ver, por ejemplo, [1]), que es el sentido de *asociación* en los diagramas de clase UML. El término "composición" se usó para clases que definen objetos componentes para sus objetos, de modo que, al destruir el objeto compuesto, estos objetos componentes también se destruyen.
2. En los diagramas de clase de UML, tanto la "agregación" como la "composición" se han definido como casos especiales de asociaciones que representan relaciones de ***parte-todo*** (que se han discutido en filosofía durante mucho tiempo). En sus definiciones, la distinción entre "agregación" y "composición" se basa en el hecho de que permite compartir una parte entre dos o más totalidades. Definen las "composiciones" como partes no compartibles (exclusivas), mientras que las "agregaciones" pueden compartir sus partes. Además, dicen algo como lo siguiente: muy a menudo, pero no en todos los casos, las composiciones vienen con una dependencia del ciclo de vida entre el todo y sus partes, de modo que las partes no pueden existir sin el todo.

Por lo tanto, aunque UML ha puesto los términos "agregación" y "composición" en el contexto correcto (de las relaciones parte-todo), no han logrado definirlos de una manera clara y sin ambigüedades, capturando las intuiciones de los desarrolladores. Sin embargo, esto no es sorprendente porque existen tantas propiedades diferentes (y matices de implementación) que pueden tener estas relaciones, y los desarrolladores no están de acuerdo en cómo implementarlas.

Y la propiedad que se suponía definía la "composición" entre los objetos OOP en la comunidad C ++ (y esta creencia todavía se mantiene ampliamente): la dependencia del ciclo de vida en tiempo de ejecución entre los dos objetos relacionados (el compuesto y su componente) es no es realmente característico de "composición" porque podemos tener tales dependencias debido a la integridad referencial también en otros tipos de asociaciones.

Por ejemplo, el siguiente patrón de código para "composición" se propuso en una respuesta SO :

```
 final class Car { private final Engine engine; Car(EngineSpecs specs) { engine = new Engine(specs); } void move() { engine.work(); } }
```

El encuestado afirmó que sería característico de la "composición" que ninguna otra clase pudiera hacer referencia / conocer el componente. Sin embargo, esto ciertamente no es cierto para todos los casos posibles de "composición". En particular, en el caso del motor de un automóvil, el fabricante del automóvil, posiblemente implementado con la ayuda de otra clase, puede tener que hacer referencia al motor para poder contactar al propietario del automóvil siempre que haya un problema.

[1] http://www.learncpp.com/cpp-tutorial/103-aggregation/

Asociación

La asociación representa la relación entre dos clases. Puede ser unidireccional (unidireccional) o bidireccional (bidireccional)

por ejemplo:

1. unidireccional

El cliente realiza pedidos

2. bidireccional

    - A está casado con B
    - B está casado con A

Agregación

La agregación es un tipo de asociación. Pero con características específicas. La agregación es la relación en una clase "completa" más grande que contiene una o más clases de "partes" más pequeñas. Por el contrario, una clase de "parte" más pequeña es una parte de una clase más grande "completa".

por ejemplo: club tiene miembros

- *Un club ("entero") está compuesto por varios miembros del club ("partes").*
- *Los miembros tienen vida fuera del club.*
- *Si el club ("completo") muriera, los miembros ("partes") no morirían con él.*
- *Porque el miembro puede pertenecer a múltiples clubes ("todo").*

Composición

Esta es una forma más fuerte de agregación. "Whole" es responsable de la creación o destrucción de sus "partes"

Por ejemplo:

Una escuela tiene departamentos

- *En este caso, la escuela ("todo") moriría, el departamento ("partes") moriría con eso. Porque cada parte puede pertenecer a un solo "todo".*

Es importante entender por qué deberíamos molestarnos en usar más de una línea de relación. La razón más obvia es describir la relación padre-hijo entre clases (cuando el padre eliminó todos sus hijos se eliminan como resultado), pero más impotentemente, queremos distinguir entre la asociación simple y la composición con el fin de colocar restricciones implícitas en la visibilidad y propagación de cambios a las clases relacionadas, una cuestión que juega un papel importante en la comprensión *y reducción de la* complejidad del sistema.

Asociación

La forma más abstracta de describir la relación estática entre clases es usar el enlace Asociación, que simplemente establece que existe algún tipo de enlace o dependencia entre dos clases o más.

Asociación débil

ClassA puede estar vinculado a ClassB para mostrar que uno de sus métodos incluye un parámetro de instancia ClassB o una instancia de devolución de ClassB.

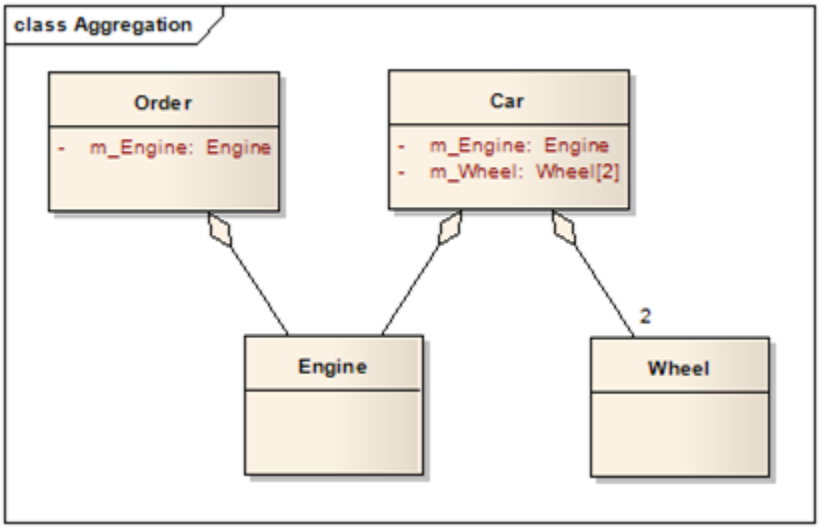
Asociación fuerte

ClassA también puede estar vinculado a ClassB para mostrar que contiene una referencia a la instancia de ClassB.

Agregación (Asociación Compartida)

En los casos donde hay una relación parcial entre ClassA (total) y ClassB (parte), podemos ser más específicos y usar el enlace de agregación en lugar del enlace de asociación, destacando que ClassB también puede agregarse mediante otras clases

en la aplicación (por lo tanto, la agregación también se conoce como asociación compartida).



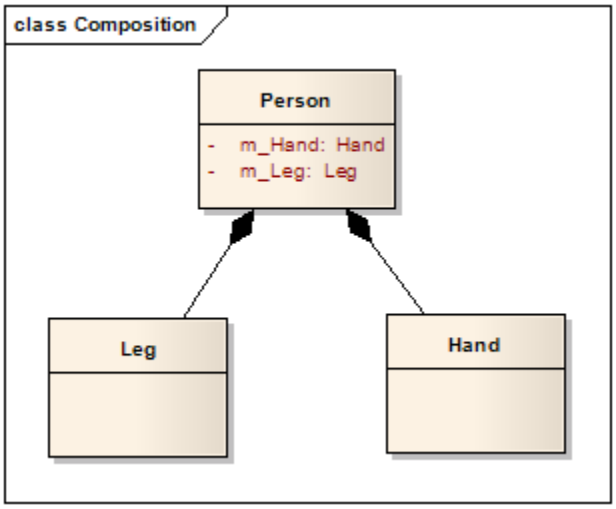
Es importante tener en cuenta que el enlace de agregación no declara de ninguna manera que ClassA posee ClassB ni que hay una relación padre-hijo (cuando el padre eliminó todos sus hijos se están eliminando como resultado) entre los dos. En realidad, ¡todo lo contrario! El enlace de agregación normalmente se utiliza para enfatizar el hecho de que ClassA no es el contenedor exclusivo de ClassB, ya que de hecho ClassB tiene otro contenedor.

Agregación vs asociación El enlace de asociación puede reemplazar el enlace de agregación en cualquier situación, mientras que la agregación no puede reemplazar la asociación en situaciones donde solo hay un 'enlace débil' entre las clases, es decir, ClassA tiene métodos que contienen parámetros de ClassB pero ClassA no mantiene la referencia a la instancia ClassB, no tiene un atributo de ClassB, solo por métodos.

Martin Fowler sugiere que el enlace de agregación no se debe utilizar en absoluto porque no tiene ningún valor añadido y altera la coherencia, citando a Jim Rumbaugh "Piense en ello como un placebo de modelado".

Composición (Asociación no compartida)

Deberíamos ser más específicos y usar el enlace de composición en los casos en que, además de la relación de parte entre ClassA y ClassB, existe una fuerte dependencia del ciclo de vida entre los dos, lo que significa que cuando ClassA se elimina, ClassB también se elimina como resultado



El enlace de composición muestra que una clase (contenedor, entero) tiene **propiedad exclusiva sobre otra clase / s (partes)**, lo que significa que el objeto contenedor y sus partes constituyen una relación padre-hijo / s.

A diferencia de la asociación y la agregación, cuando se usa la relación de composición, la clase compuesta no puede aparecer como un tipo de retorno o tipo de parámetro de la clase compuesta. Por tanto, los cambios en la clase compuesta no se pueden propagar al resto del sistema. En consecuencia, el uso de composición limita el crecimiento de la complejidad a medida que crece el sistema.

Midiendo la complejidad del sistema

La complejidad del sistema se puede medir simplemente mirando un diagtwig de clase UML y evaluando las líneas de relación de asociación, agregación y composición. La forma de medir la complejidad es determinar cuántas clases pueden verse afectadas al cambiar una clase en particular. Si la clase A expone la clase B, cualquier clase dada que use la clase A puede teóricamente verse afectada por los cambios a la clase B. La sum de la cantidad de clases potencialmente afectadas para cada clase en el sistema es la complejidad total del sistema.

Puede leer más aquí: http://aviadezra.blogspot.com/2009/05/uml-association-aggregation-composition.html

---

- Una relación entre dos objetos se denomina **asociación**.
- Una asociación se conoce como **composición** cuando un objeto posee otro.
- Mientras que una asociación se conoce como **agregación** cuando un objeto contiene a otro objeto.

El problema con estas respuestas es que son la mitad de la historia: explican que la agregación y la composición son formas de asociación, pero no dicen si es posible que una asociación no sea ninguna de ellas.

De acuerdo con algunas breves lecturas de muchos posts sobre SO y algunos documentos de UML, deduzco que existen 4 formas principales de asociación de clases:

1. composición: A está compuesta por B; B no existe sin A, como una habitación en un hogar
2. agregación: A tiene-a B; B puede existir sin A, como un estudiante en un aula
3. dependencia: A usa-a B; no hay dependencia del ciclo de vida entre A y B, sino un parámetro de llamada de método, valor de retorno o un temporal creado durante una llamada de método
4. generalización: A es-un B

Cuando una relación entre dos entidades no es una de ellas, simplemente puede llamarse "una asociación" en el sentido genérico del término, y describir otras formas (nota, estereotipo, etc.).

Supongo que la "asociación genérica" está destinada a ser utilizada principalmente en dos circunstancias:

- cuando los detalles de una relación aún se están resolviendo; tal relación en un diagrama se debe convertir tan pronto como sea posible a lo que realmente es / será (uno de los otros 4).
- cuando una relación no concuerda con ninguno de los 4 predeterminados por UML; la asociación "genérica" todavía le da una forma de representar una relación que "no es una de las otras", para que no se quede atrapado utilizando una relación incorrecta con una nota "esto no es realmente agregación, es solo que UML no tiene ningún otro símbolo que podamos usar "

Creo que este enlace hará su tarea: http://ootips.org/uml-hasa.html

Para entender los términos, recuerdo un ejemplo en mis primeros días de programación:

Si tiene un objeto 'tablero de ajedrez' que contiene objetos 'cuadros' que es **composición** porque si se elimina el 'tablero de ajedrez' ya no hay motivo para que los cuadros ya existan.

Si tiene un objeto 'cuadro' que tiene un objeto 'pieza', y la pieza se la comen, el objeto 'cuadro' aún puede existir, eso es **agregación**

- Otros Ejemplos de **asociaciones**, con diferencias de tipo conceptual:

Composición: Aquí es donde una vez que destruyas un objeto (Escuela), otro objeto (Aulas) que está unido a él también será destruido. Ambos no pueden existir de forma independiente.

Agregación: Esto es exactamente lo contrario de la asociación anterior (`Composition`) donde una vez que matas un objeto (`Company`), el otro objeto (`Employees`) que está ligado a él puede existir por sí mismo.

La composición y la agregación son las dos formas de asociación.

Composición (Si elimina "todo", "parte" también se elimina automáticamente – "Propiedad")

- Crea objetos de tu clase existente dentro de la nueva clase. Esto se llama composición porque la nueva clase se compone de objetos de clases existentes.
- Por lo general, usa variables de miembros normales.
- Puede usar valores de puntero si la clase de composición maneja automáticamente la asignación / desasignación responsable de la creación / destrucción de subclases.

Figure 1 : Composition

Composición en C ++

```
 #include     using   namespace   std;   /*********************   Engine   Class
*****************/  class  Engine  {  int  nEngineNumber;  public:  Engine(int
nEngineNo); ~Engine(void); }; Engine::Engine(int nEngineNo) { cout<<" Engine ::
Constructor " <<<"="" engine="" ::="" destructor="" "="" <<<"----------------=""
```

```
start="" of="" program="" --------------------"<<<"-------------="" inside="" block="" ------------------"<<="" iostream="" style="box-sizing: border-box;">
```

Salida

```
 --------------- Start Of Program -------------------- ------------- Inside Car Block ------------------ Engine :: Constructor Car :: Constructor Car :: Destructor Engine :: Destructor ------------- Out of Car Block ------------------ ------------- Inside Bus Block ------------------ Engine :: Constructor Bus :: Constructor Bus :: Destructor Engine :: Destructor ------------- Out of Bus Block ------------------ --------------- End Of Program --------------------
```

Agregación (Si elimina "todo", puede existir "Parte" - "No Propiedad")

- Una agregación es un tipo específico de composición en la que no está implícita la propiedad entre el objeto complejo y los subobjetos. Cuando se destruye un agregado, los subobjetos no se destruyen.
- Normalmente, utiliza variables de puntero / variable de referencia que apuntan a un objeto que vive fuera del scope de la clase agregada
- Puede usar valores de referencia que apuntan a un objeto que vive fuera del scope de la clase agregada
- No somos responsables de crear / destruir subclases

Figure 2: Aggregation

Código de agregación en C ++

```
 #include  #include  using namespace std; /********************** Teacher Class *****************/  class  Teacher  {  private:  string  m_strName;  public:  Teacher(string  strName);  ~Teacher(void);  string  GetName();  };  Teacher::Teacher(string  strName)  :  m_strName(strName)  {  cout<<"  Teacher  ::  Constructor --- Teacher Name :: "<<m_strname<<<"="" teacher="" ::="" destructor="" ---="" name="" "<<m_strname<<<"-------------="" inside="" block="" ----------="" -----"<<="" style="box-sizing: border-box;"></m_strname<
```

Salida

```
 --------------- Start Of Program -------------------- Teacher :: Constructor --- Teacher Name :: Reference Teacher Teacher :: Constructor --- Teacher Name :: Pointer Teacher ------------- Inside Block ------------------ Department :: Constructor Department :: Destructor ------------- Out of Block ---------- Teacher :: Destructor --- Teacher Name :: Pointer Teacher Teacher :: Destructor
```

```
--- Teacher Name :: Reference Teacher --------------- End Of Program ----------
---------
```

Me gustaría ilustrar cómo se implementan los tres términos en Rails. ActiveRecord llama a cualquier tipo de relación entre dos modelos una `association`. No se encontrarán muy a menudo los términos `composition` y `aggregation`, al leer documentación o artículos, relacionados con ActiveRecord. Se crea una asociación al agregar una de las macros de clase de asociación al cuerpo de la clase. Algunas de estas macros son `belongs_to`, `has_one`, `has_many`, etc.

Si queremos configurar una `composition` o `aggregation`, necesitamos agregar `belongs_to` al modelo propiedad (también llamado child) y `has_one` o `has_many` al modelo propietario (también llamado parent). Si configuramos la `composition` o `aggregation` depende de las opciones que pasamos a la llamada `belongs_to` en el modelo hijo. Antes de Rails 5, la configuración de `belongs_to` sin ninguna opción creó una `aggregation`, el elemento secundario podría existir sin un elemento primario. Si queríamos una `composition`, necesitábamos declararla explícitamente agregando la opción `required: true`:

```
class Room < ActiveRecord::Base belongs_to :house, required: true end
```

En Rails 5 esto fue cambiado. Ahora, al declarar que una asociación `belongs_to` crea una `composition` de forma predeterminada, el elemento secundario no puede existir sin un elemento primario. Por lo tanto, el ejemplo anterior se puede volver a escribir como:

```
class Room < ApplicationRecord belongs_to :house end
```

Si queremos permitir que el objeto hijo exista sin un padre, debemos declararlo explícitamente a través de la opción `optional`

```
class Product < ApplicationRecord belongs_to :category, optional: true end
```