Design Codes

Aviad Ezra on Software Architecture

Thursday, May 28, 2009

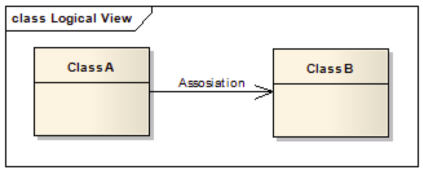
UML Class Diagram: Association, Aggregation and Composition

The UML Class diagram is used to visually describe the problem domain in terms of types of objects (classes) related to each other in different ways.

There are 3 primary inter-object relationships**:** ***Association***, ***Aggregation***, and ***Composition****.* Using the right relationship line is important for placing implicit restrictions on the visibility and propagation of changes to the related classes, a matter which plays an important role in understanding and reducing system complexity.

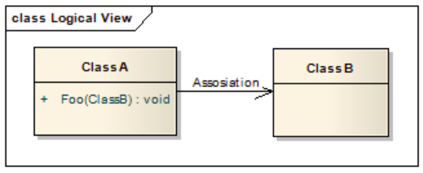
Association

The most abstract way to describe static relationship between classes is using the **Association** link, which simply states that there is some kind of a link or a dependency between two classes or more.



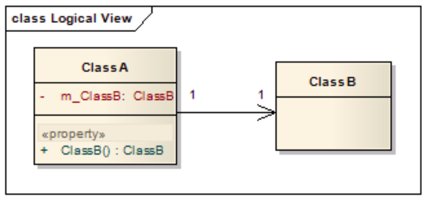
Weak Association

ClassA may be linked to ClassB in order to show that one of its methods includes parameter of ClassB instance, or returns instance of ClassB.



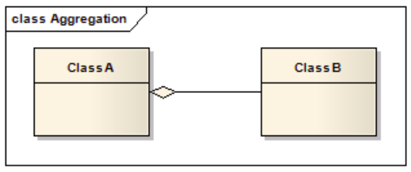
Strong Association

ClassA may also be linked to ClassB in order to show that it holds a reference to ClassB instance.



Aggregation (Shared Association)

In cases where there's a part-of relationship between ClassA (whole) and ClassB (part), we can be more specific and use the aggregation link instead of the association link, highlighting that the same ClassB instance can also be aggregated by other classes in the application (therefore aggregation is also known as shared association).



It's important to note that the aggregation link **doesn't state** in any way that ClassA owns ClassB **nor** that there's a parent-child relationship (when parent deleted all its child's are being deleted as a result) between the two. Actually, quite the opposite! The aggregation link is usually used to stress the point that ClassA instance is not the exclusive container of ClassB instance, as in fact the same ClassB instance has another container/s.

Highlights!

Scale up and scale out with .NET and Azure

Association, Aggregation and Composition

The Pillars of Concurrency

Scaling Up with STM.NET (Software Transactional Memory)

Component Testability KILLERS (and suggested solutions)

Testing in Production – Benefits, Risks and Mitigations

Software Interview Nightmares

Aviad Ezra

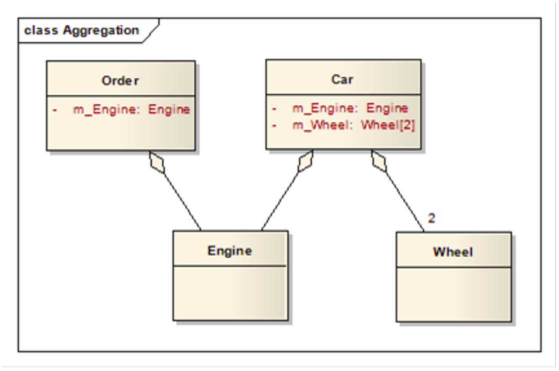
About me

Software developer at Microsoft with over 17 years of experience building large, distributed software systems for the cloud and on-premises.

Design Patterns (10) MEF (2) Multi Threading (7) MVC (3) MVP (4) MVVM (4) MVXX (7) Networking (7) Silverlight (3) Testing (2) UML (6)

Blog Archive

- ► 2017 (1)
- ► 2016 (1)
- ► 2015 (1)
- ► 2014 (4)
- ► 2013 (1)
- ► 2012 (1)
- ► 2011 (2)
- ► 2010 (7)



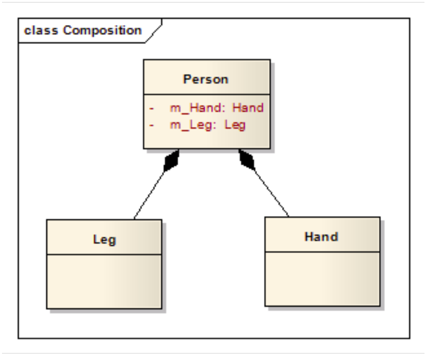
Aggregation v.s. Association

The association link can replace the aggregation link in every situation, while aggregation cannot replace association in situations where there's only a 'weak link' between the classes, i.e. ClassA has method/s that contain parameter of ClassB, but ClassA doesn't hold reference to ClassB instance.

> *Martin Fowler suggest that the aggregation link should not be used at all because it has no added value and it disturb consistency, Quoting Jim Rumbaugh "Think of it as a modeling placebo".*

Composition (Not-Shared Association)

We should be more specific and use the composition link in cases where in addition to the part-of relationship between ClassA and ClassB - there's a strong lifecycle dependency between the two, meaning that when ClassA is deleted then ClassB is also deleted as a result



The composition link shows that a class (container, whole) has exclusive ownership over other class/s (parts), meaning that the container object and its parts constitute a parent-child/s relationship.

Unlike association and aggregation, when using the composition relationship, the composed class cannot appear as a return type or parameter type of the composite class. Thus, changes to the composed class cannot propagate to the rest of the system. Consequently, usage of composition limits complexity growth as the system grows.

Clarification: It is possible for a class to be composed by more than one class. For example, ClassA may be composed by ClassB and ClassC. However, unlike aggregation, instances of ClassB and ClassC will never share **the same** ClassA instance. That would violate the *propagation of changes* principle. ClassB instance will have its own instance of ClassA, and ClassC instance will have its own instance of ClassA.

Measuring system complexity

System complexity can be measured simply by looking at a UML class diagram and evaluating the association, aggregation, and composition relationship lines. The way to measure complexity is to determine how many classes can be affected by changing a particular class. If class *A* exposes class *B*, then any given class that uses class *A* can theoretically be affected by changes to class *B*. The sum of the number of potentially affected classes for every class in the system is the total system complexity.

Recommended Book on UML

You should read Martin Fowler's book: UML Distilled: A Brief Guide to the Standard Object Modeling Language (3rd Edition). It's pure gold.

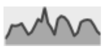
I am also answering questions on Quora @https://www.quora.com/profile/Aviad-Ezra

My other blog: Interview Questions (and answers) by Aviad Ezra

Posted by Aviad Ezra at 5:27 AM

Labels: UML

Pageviews last month



15,568

86 comments:

Anonymous July 22, 2009 at 11:59 AM

This is a good explaination of relationships among classes.
Recently I have to make a presentation file about this topic, may I use the content of this article ?

This is Jack Yeh (jao_chin@so-net.net.tw)

Reply

aviade July 22, 2009 at 6:16 PM

Sure, Feel free to use it :-)

Reply

Anonymous July 23, 2009 at 5:56 PM

Thank you very much.

Jack Yeh

Reply

dGrouch August 6, 2009 at 2:54 PM

Been all over looking for the best explanations about these relationships. This is by far the best. Simple, short and direct to the point. Thanks man.

Reply

aviade August 9, 2009 at 11:57 AM

Thanks a lot, I appreciate the feedback :-)

Reply

Anonymous August 24, 2009 at 1:18 PM

Excellent summary, finally after may search I could find out how to draw the differece between strong association and weak association.

Reply

Anonymous October 22, 2009 at 8:24 AM

Nice explaination.

thanks.

Reply

Hari February 1, 2010 at 9:41 PM

Great explanation..

Thanks

Reply

Anachronox February 3, 2010 at 9:06 PM

Thanks it was really nice . Specially the optional aggregation thing

Reply

Anonymous February 6, 2010 at 1:22 PM

nice helped a lot

Reply

Anonymous February 23, 2010 at 7:53 PM

are you sure that v can have 2 whole(s) in a whole-part relationship for a given part???
eg:A(whole)------->B(part) and
C(whole)-------->B(part)

because v hav been taught that it is not possible..
plz send a response ASAP to
ruma.bhatnagar@gmail.com

Reply

Anonymous February 28, 2010 at 5:51 PM

Finally a good explanation! Thanks so much.

Reply

rajesh July 20, 2010 at 1:22 PM

Good explanation, i read many books but did not find explanttion like this. keep it up.

Reply

vrsanaidu August 7, 2010 at 11:15 AM

Excellent work, finally understood relations ship between these 3 definitions

Reply

Anonymous August 18, 2010 at 10:48 AM

great work...it 'll be still be greater if u can pls put Java coding examples for each one.

Reply

yanny August 19, 2010 at 1:25 PM

This comment has been removed by a blog administrator.

Reply

ejaz December 17, 2010 at 3:43 AM

Thanx...really v nice stuff...for basic information and advance information as well

Reply

Shabu February 28, 2011 at 3:22 AM

Excellent!! All UML books are confusing the above concepts. But this one simply describes it. Congrats!!

Reply

G.Vivek Venkatesh April 21, 2011 at 7:41 AM

Thanks a lot...This is the best explanation I have seen for this topic...:)

Reply

John McLaren April 26, 2011 at 9:39 PM

finally a reasonable explaination of the uml and those arrows

Reply

Bensi April 26, 2011 at 9:41 PM

Straight forward.. finally I understood UML !!! wait what's aggregation? lol

Reply

Kelso April 26, 2011 at 9:45 PM

it would have better if completed this explaination with some code examples like which class holds what..

Reply

Anonymous May 30, 2011 at 4:22 AM

Thanks for a very accurate and simplified explanation on this topic.

Reply

Anonymous August 1, 2011 at 1:27 PM

Thanks for your explanation, it is much clearar compared to all other sources I read. However, one detail is still confusing. You mention in your explanation of aggregation and composition that one class is or is not exclusive owner of another class. Did you rather mean instances of a specific class? I guess class A and class B can have a composite relationship to class C, if class A and Class B have different instances of class C.

Reply

aviade August 3, 2011 at 7:29 AM

Yes, that's correct. Multiple classes can have a composite relationship with the same class, as long as they don't expose the composed class to the outside world and there's a life-cycle dependency between them.

Reply

Anonymous August 31, 2011 at 7:10 AM

It is a straightforward explanation. But does it mean that aggregation is a stronger case of association.

Reply

Anonymous October 31, 2011 at 9:19 AM

Yes , it's excellent explanation.

Thanks,
Anji

Reply

Anonymous November 25, 2011 at 1:17 PM

Thanks..
Excellent post

Reply

Anonymous December 3, 2011 at 4:21 AM

Dude you are a star !!

Reply

Hikari December 6, 2011 at 10:12 AM

Really a great explanation, I've never really understood the meaning/semantic of them and where to use them, and u put it very simple!

I still have 2 situations I dodn't understand. So, when using composite, it means an object of class A (the one with the black symbol) stores 1 or more objects of class B, and when that A object is deleted, those B objects are also deleted?

Another question. You talked about weak aggregation where class A has operation that receives parameter or returns object of class B. But what if an operation of class A does some processing using class B (being it static class or object instance)? That operation could call a class C operation, receive a class B object return, use that object internally but its return doesn't involve class B anymore. So, if inside class A's operation there's a use of class B (and therefore if class B is missing we can't compile class A!), will we need to add assossiation from A to B?

To finish the comment, I'd add the information that when we have an assossiation of one-to-many, it will be implemented by adding to class A a collection of class B's objects. It's facultative to explicity put in class A the attribute that will hold that collection or leave it implicit by the assossiation.

I took some time to learn how to implement this kind of model, now I myself like to explicitly show all classes attributes, and leave multiplicity to inform how a class knows about the other. In domain models I use multiplicity to inform how one depends of the other, in database models I use to inform their relational assossiations, and in class models I use to inform if a class relies on the other (it knows about that class and won't compile if the class is missing).

Reply

Anonymous December 12, 2011 at 4:56 AM

Now I know UML.

Reply

aviade December 12, 2011 at 5:47 AM

Hikari, the answer for your 1st question is yes - in composition, when the container class is deleted, all the contained objects are deleted as a results.
I didn't quite understand your second question. My guess is that you were referring to a case where Class A is weakly associated with Class B, and in Class A internal implementation it get instance of Class C through Class B. In that case, there's a weak association relationship between Class A and Class C - and that should made visible in the UML diagram.

Hope that helps. Aviad

Reply

Anonymous January 15, 2012 at 10:02 PM

This proved to be of a great help for my Automotive Software Engg. exam. Thanks a ton!!!
Santosh G.

Reply

Almantas January 19, 2012 at 1:00 PM

Superb explanation! Have one question regarding association vs aggression. In your example of strong association, you showed its multiplicity 1:1. That means both classes have class members as pointers to each other class. What about 1:* (one to many) association multiplicity? Would it be same as aggregation? Sometimes, I see 0..1:1..*(zero or one to one or many) multiplicity when looking at some relational database models (drawn with UML class diagram) where associations are used to describe relations between parent and child tables (for example, there are tools that allow to generated database schema from class diagram). But if one-to-many association is the same as aggregation then there's a logic conflict: in relation tables, a child table record can hold a reference (parent_id column) to only one parent record. So those associations are more composition style relations. Any thoughts on that?

Reply

Almantas January 19, 2012 at 1:22 PM

Some additional notes: aggregation might be considered as many-to-many association relation because both end classes might be used in different related instances, so in some situations, it might be no matter what class is composite and what component - both might be included in each other. On the other hand, one-to-many association might be considered as composition .... but nop, because association does not restrict the same child object be assigned with another parent object. Actually, I do very agree with Martin Fowler about aggregation link should not be used at all because it adds ambiguity, because the same might be achieved with association with multiplicity one-to-many.

Reply

aviade January 19, 2012 at 4:09 PM

The multiplicity in strong association is a mistake - thanks for bringing this up!

You can have one to many multiplicity for all the link types - Association, aggregation and Composition.
Many to many is possible only for association, as whole-part (or parent-child) can only go one way.
You also need to remember that with composition, the composed class is only visible to its parent.

Reply

Almantas January 19, 2012 at 11:37 PM

I would like to discuss your answers further :)

a) "The multiplicity in strong association is a mistake" - So, we don't need to put 1:1 multiplicity when modeling strong associations at all? If so then both strong and weak associations will look the same (both will just have association without any multiplicity). How to distinguish them? Maybe I'v missed something in your answer...

b) "You can have one to many multiplicity for all the link types" - why do we need to have that on aggregation and Composition? Composition is for modeling one-to-many relation, and the same is with aggregation.

c) I would like to argue about "Many to many is possible only for association". IMHO aggregation is very similar to many-to-many relation. For example, assume there's an aggregation relation between "part" and "airplane" classes . Then both classes might consist from each other: one part might belong to several airplanes, and airplane might consist of several parts. In relational database wold, that is definite a many-to-many relation. Though maybe we should use many-to-many association in such situations ... ?

Reply

aviade January 20, 2012 at 1:21 AM

What I meant by "multiplicity in strong association is a mistake" is that I shouldn't have drawn 1-1 multiplicity because it's confusing. You can have 1-1 multiplicity in strong association, but it's not a must.

Here's my understanding,
In aggregation and composition - you can have 1-1 and 1-many multiplicity.
In association (week and strong) - you can have 1-1, 1-many and many-many multiplicity.

Reply

Anonymous February 10, 2012 at 4:11 AM

Good explanation this helped me to understand the class relationship well.

Reply

Anonymous March 24, 2012 at 9:58 AM

Very good explanation.Was of great of help in understanding the concepts

Reply

Anonymous April 4, 2012 at 8:44 AM

thanks...

Reply

Anonymous April 5, 2012 at 9:03 PM

Excellent and Superb Explanation!!!

Reply

Purush April 5, 2012 at 9:10 PM

Very good explanation Superb!!!! Great Job

Reply

Purush April 5, 2012 at 9:11 PM

Very good Explanation

Reply

youtube html5 player April 9, 2012 at 4:14 PM

Thanks for sharing your info. I really appreciate your efforts and I will be waiting for your further write ups thanks once again.

Reply

Vipin Nair June 13, 2012 at 6:29 AM

Simply great...

Reply

Anonymous June 27, 2012 at 6:29 AM

Keeping it simple always works!!! Good one.

Reply

gschen August 28, 2012 at 6:38 AM

Excellent explaination.

Reply

Shadeven November 17, 2012 at 12:35 AM

Good stuff

Reply

Shadeven November 17, 2012 at 12:35 AM

Good stuff!

Reply

Danish Khan December 30, 2012 at 12:02 PM

Quite helpful stuff. Thanks

Reply

Aniruddha January 15, 2013 at 11:20 PM

All these days I could not understand whether to use association or aggregation! you opened my eyes kudos to you.
Keep up your good work!

Reply

Sumesh.S.G January 22, 2013 at 8:43 AM

Thanks for sharing this outstanding stuff.

Reply

Dedunu Dhananjaya February 27, 2013 at 12:31 PM

This post helped me to get ready to exam thanks buddy!!

Reply

uzumaki May 9, 2013 at 2:14 PM

This the best explanation I have ever read about the subject !

Thank you soooooo much :D

Reply

Jaafar Talik May 30, 2013 at 4:09 AM

Thanks for sharing :) I have just one question,
In the post,as I understood, the only difference between Aggregation and Association, technically, is that the Aggregation can't model the Association(case of weak Association), Actually what mentionned as "weak Association" is the UML "Dependency" relation, so if that's the case what's the difference between the Aggregation and Association?

Reply

Replies

Rich April 10, 2017 at 12:08 AM

This is a great point others have missed. The "Weak Association" is actually a "Dependency" in UML. The difference between Aggregation and Association is that aggregation uses a conceptual "is part of" relationship. Like a car is made of an engine and doors. If there is no whole-part relationship is just an association. An aggregation is a specific case of association.

Reply

prasanth May 30, 2013 at 7:24 AM

one of the best explanations ...thank you

Reply

Vladimir Kishlaly August 14, 2013 at 2:24 PM

I've been confusing these meanings for years.. Now all is clear. Thanks!

Reply

Rizwan Ishtiaq September 10, 2013 at 8:35 AM

Dear aviade,

Such a nice explanation. i have to question

i class A have some public private or protected methods, in which we are creating the objects of other classes (let class B or C)
Then what relationship this will denote?

Rizwan Ishtiaq

Reply

Replies

aviade September 11, 2013 at 8:13 AM

if class B and C are not exposed to other classes in the application, then the relationship is composition. The scope of the methods within which the classes was created makes no difference.

Hope that helps

Reply

Sophist110 November 22, 2013 at 2:44 PM

Simple and Effective explanation. good job

Reply

Mbulungo Musetsho March 23, 2014 at 8:53 PM

i like this explanation

Reply

Developer April 28, 2014 at 8:12 PM

Good explanation, should have higher page rank on Google search.

Reply

Carter Dunley May 22, 2014 at 10:54 PM

Thanks for the great explanation the three primary relationships. I like how you took it and explained it even more. I was still looking up What is a Class Diagram in UML and found this to be pretty helpful as well. Thanks for all of your great work!

Reply

Rahul June 4, 2014 at 10:53 AM

Aggregation and Composition both are specialized form of Association.
Composition is again specialize form of Aggregation.

http://www.javabench.in/2011/08/difference-between-association.html

Reply

Bruce Li June 5, 2014 at 7:10 AM

Hello, thank you very much for your great article, but I still don't understand the arrow directions. In Association, arrow points from Class A -> Class B, however, in Aggregation, the diamond (which I, who may may be wrong, think is another kind of arrow) points from Class B to Class A. Would you be kindly to explain it to me? How can I decide which direction should arrow / diamond point to when I draw a class diagram?

Reply

Palak Maheria June 25, 2014 at 9:46 AM

This explanation is simple but very effective. It is easily understandable. Great job

Reply

Santo October 17, 2014 at 5:47 AM

Great explanation.. Thanks

Reply

Gabrielius Liaškus February 3, 2015 at 4:43 PM

I guess in Composition you can still return a Composed class-type object in Composite class, when it's read-only or value type? Or Composition strictly says that other objects than Composite class cannot use and see Composed class?

Reply

indhu u February 17, 2015 at 6:21 AM

good explanation and give with example........

Reply

Joyanta Sen May 9, 2015 at 8:53 PM

As you mentioned that A and class B can have a composite relationship to class C, if class A and Class B have different instances of class C and as long as they don't expose the composed class to the outside world and there's a life-cycle dependency between them.
So if they expose Class C to the other class even though it is created inside Class A or B, then the relationship will be Aggregation ?

Reply

Replies

aviade April 25, 2016 at 6:34 AM

Correct. If ClassC is exposed the composition link cannot be used. Exposing an object (via public property) that is deleted by its container is also error prone, as it increases the risk for NULL reference exceptions.

Reply

Shalin Siriwaradhana June 17, 2015 at 9:10 AM

Good explanation. it would have been better if there were many examples.

Regards,
creately

Reply

Amit Rastogi July 22, 2015 at 12:50 PM

Good explanation. Thanks.

Reply

Ramu Legend August 8, 2015 at 3:14 PM

Hi Aviade,
Unlike association and aggregation, in the composition relationship, the composed class cannot appear as a return type or parameter type of the composite class, thus changes in the composed class cannot be propagated to the rest of the system.

Could you please explain the above statement.

Reply

Replies

aviade April 25, 2016 at 6:25 AM

Say that ClassA (composite) has composition relationship with Class B (composed). And ClassC is using ClassA.

ClassA cannot have a public property that exposes ClassB. Thus, ClassC cannot see or use ClassB through ClassA. Which means that changes to ClassB cannot affect (or change functionality of) ClassC.

Reply

Muruges Krishnan October 10, 2015 at 10:08 PM

Great explanation. really helped me to understand the differences between these concepts.

Reply

David Zagi November 17, 2015 at 8:26 AM

Really good explanation. Thank

Reply

Arvind Kumar Avinash November 22, 2015 at 4:06 PM

Good work! Keep it up

Reply

Marshal truman April 24, 2016 at 8:25 AM

Very much useful.. Simple and precise..

Reply

Unknown December 21, 2016 at 3:46 PM

Best explanation I've found so far. Thank you!

Reply

Vishal Sadaphal February 13, 2017 at 5:43 AM

This is the best and most accurate explanation i could find on internet. Thank you for putting this together. It would be great if you can give some examples on Class Diagrams and ideal design for them.

Reply

Xabier Montez May 26, 2017 at 5:21 AM

Hi Aviad,
First, thanks a bunch for this useful article, just one doubt about your example of Aggregation: if Wheel and Engine objects were exclusive of Car object, then would it be a Composition relationship?
This doubt arises because in some topics about Composition relationships almost always people give a Car-Engine-Wheel example as that kind of relationship.
Maybe, all depends about the requirements to consider if Car-Engine-Wheel example would be Aggregation or Composition relationship.

Thanks in advance,
Xabier

Reply

Unknown May 28, 2017 at 5:13 AM

Agreed, so many bad examples out there. Thanks for the clarity!

Reply

Mohammed Suhail October 5, 2017 at 7:37 AM

Thank you

Reply

Enter your comment...

Comment as: Eyal Alon (Google **Sign out**

Publish **Preview** ☐ Notify me

Links to this post

Create a Link

Newer Post Home Older Post

Subscribe to: Post Comments (Atom)