Three Patterns in Object Modeling

Paul Asman, Federal Reserve Bank of New York, July 1999

James Rumbaugh has written, "Basically, attributes are for representing relationships between objects and values without identity, whereas associations are for representing relationships between objects and other objects" (Rumbaugh 1996). When you apply this principle, though, you soon encounter difficult cases. It is clear that a character is a value without identity. But this is less clear for a string, which seems to be both without identity as well as one of the "other objects," an instance of class String. This is even less clear for an address, although this is one of Rumbaugh's examples of an attribute. Perhaps an address can be a value without identity, for example in domains where addresses are used only for generating mailing labels. But an address can also be a first-class object of its own, as it would be in an insurance application.

Distinguishing between attributes and associations is a problem for modeling, not implementation. When you implement a model in an object-oriented language, each component of that model is either omitted or made part of a class specification. There is no place for anything else: the code is not sand, and you cannot draw a line representing an association in it. Nor is there a need for anything else: code must be accurate and efficient, not expressive.

Models must also be accurate, but they must be expressive as well. You could model all associations as attributes from the beginning, just as you could model all attributes as associations. (For the first option, see Velho and Carapuça 1994; for the second, see Tanzer 1995, who rejects it.) The choice between using an attribute or an association therefore cannot be decided by accuracy, that is, by reference to the reality underlying the model. Instead, you choose one or the other for its expressive power.

Determining what to model as an association and what as an attribute therefore is an art, not a science. However, you can subject aspects of this determination to science, or at least to engineering. These patterns attempt to take the broad heuristics for identifying attributes and associations found in object modeling literature, and create from them more mechanical decision procedures. Successful application of the patterns will reserve art for what is beyond science.

Since the models created in analysis and design serve different expressive needs, there are different patterns to apply. The first pattern, *Deferred Attributes*, addresses forces that apply both to analysis and design. The second, *Associations for Analysis*, addresses forces specific to analysis, and the third, *Attributes for Design*, addresses forces specific to design.

All three patterns apply only when there is some question whether to model with an attribute or an association. The question does not arise with every candidate association. It does not arise, for example, when modeling the relationship between two robust

domain objects. It also does not arise when creating association objects, which by definition “associat[e] two other objects” (Boyd 1998).

References:

- Boyd, Lorraine L., “Business Patterns of Association Objects,” in Martin, Robert C., Dirk Riehle, and Frank Buschmann, *Pattern Languages of Program Design 3*, Addison-Wesley, 1998.
- Rumbaugh, James, "A Search for Values: Attributes and Associations," *Journal of Object-Oriented Programming*, Volume 9, Number 3, June, 1996, pp. 6-8,49.
- Tanzer, Christian, "Remarks on Object-Oriented Modeling of Associations," *Journal of Object-Oriented Programming*, Volume 7, Number 9, February, 1995, pp. 43-46.
- Velho, Amândio Vaz and Rogério Carapuça, "From Entity-Relationship Models to Role-Attribute Models," in *Proceedings of the 12th International Conference on Entity-Relationship Approach, Arlington, Dallas, USA, December, 1993*, Springer-Verlag, 1994, and "Attribute: A Semantic and Seamless Construct," in Magnusson, Boris, Bertrand Meyer, and Jean-Marc Nerson, *Technology of Object-Oriented Languages and Systems: Proceedings of the Thirteenth International Conference Tools Europe '94 Versailles, France*, Prentice-Hall, 1994.

Name

Deferred Attributes

Context

You are creating a model. You know that any associations you draw will eventually be implemented as attributes (or not at all), but you also know that your model will lose its expressive power if it does not show links between classes. You know as well that you can go too far in drawing associations, cluttering your model with classes that are not an interesting part of your domain.

This pattern covers cases where the forces are the same whether you are engaged in analysis or design. Other patterns, *Associations for Analysis* and *Attributes for Design*, respond to the forces specific to those contexts.

Problem

Which eventual attributes are best modeled as associations, whether in analysis or design?

Forces

- Putting associations into a model clearly shows "the web that ties an entire model together" (Rumbaugh 1996); without associations, you don't see this web.
- An "enormous number of associations of very different importance" (Tanzer 1995) produces clutter and confusion, and obscures the relevant portion of the web.
- Elements have different roles in different domains. For example, a billing domain is likely to use postal code only for generating mailing labels; there will be little to say about postal code other than what it is. An insurance domain, however, is likely to use postal code for setting rates, and may associate it with risk level.
- Elements have connections with different directionality in different domains. (See Papurt 1994.) A real estate tax domain, for example, requires access to property owners through addresses as well as access to addresses through property owners, while most other domains require only the latter.
- Objects hold some elements throughout their existence or nearly so. A person, for example, generally receives a name shortly after birth and has some name from that time on. Objects hold other elements temporarily, or only at certain times. A person owns cars, for example, but generally not when a toddler. (See D'Souza 1994.)
- Different assumptions are made about the visibility of elements when they are modeled as attributes or associations. Rational Rose, for example, makes attributes private and associations public by default. Unless a modeler changes visibility, then, Rose generates Java code implementing attributes as private attributes and associations as public attributes.

Solution

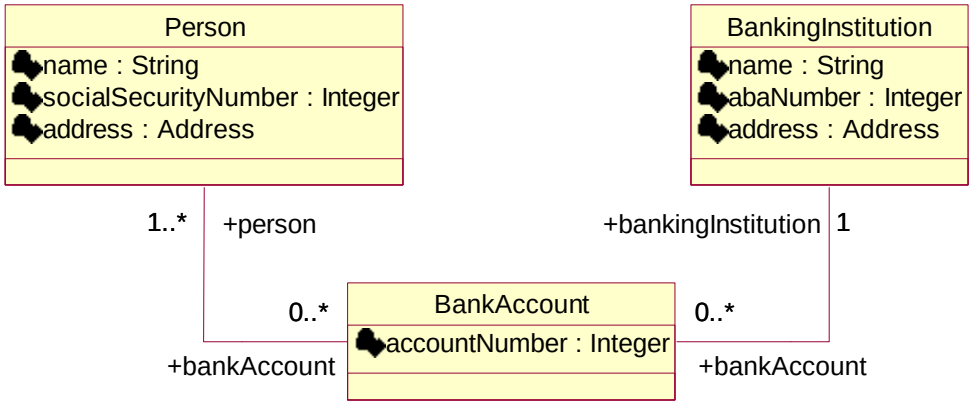
1. If there is nothing to say about a simple element other than what it is, and nothing you will ask it to do, model it as an attribute. An example is character, of which an example is the letter 'c'.
2. If everything that you will ask an element to do comes as part of any object-oriented language you might use, model it as an attribute. Examples include strings and numbers. You may ask an element that is an instance of String to return its first character, but the code for this will come with the implementation language; you won't need to write it.
3. You should normally treat the fundamental types you create (such as currency; see Fowler 1997) as if they came with the language. However, since you must code these fundamental types, you should represent them as objects with operations on at least one diagram in a model. A diagram dedicated to such fundamental types would be appropriate. Associations will not be important on this diagram.
4. If only one class has access to an element, you should normally model it as an attribute of that class. Most identifiers (e.g. name, social security number, and bank routing number) fall under this guideline. Since these elements often have methods defined for such operations as formatting and validation, though, they should be represented as objects on at least one diagram in a model. (Bank routing

numbers, for example, are validated by the computation of the last digit, which is a check digit.) A diagram dedicated to such elements would be appropriate. Associations will not be important on this diagram.

5. If more than one class has access to an element that is not subject to the previous guidelines, and if different classes in a model use this element differently, you should normally model it with associations. An example is bank account, which person and banking institution use differently. Note the qualification "in a model," which allows you to model an element differently in different domains. (An element accessible from multiple classes in one domain may be accessible only from one class in another.)
6. If you have identified a bi-directional relationship between two classes, model it as an association. (You need not model it as a bi-directional association, though: see *Attributes for Design* solution item 1.) Do not model it by specifying an attribute in each class of the type of the other class.
7. When objects do not hold elements throughout their lifetimes (or nearly so), you should normally model those elements with associations rather than attributes. This is especially important if you think you may subclass these objects, for it is generally better to redraw associations to a subclass than to carry along or reassign inappropriate attributes. Note that this guideline offers no guidance when elements are held throughout the lifetime of an object (as are, for example, one's parents).

Example

The following sketch shows a simplified part of a banking domain modeled according to the solution. Three classes are used as attribute types rather than represented as classes linked by associations: String and Integer, which are Java classes, and Address, which is not.



(created in UML with Rational Rose 98i)

Resulting Context

While this pattern solves many of the issues in choosing between attributes and associations, it does not recognize the different expressive goals of analysis and design

models. If you are creating an analysis model, you should apply *Associations for Analysis* simultaneously to or after applying this pattern. If you are creating a design model, you should apply *Attributes for Design* in the same way.

References

- D'Souza, Desmond, "Working with OMT, part 2," *Journal of Object-Oriented Programming*, Volume 6, Number 9, February 1994, pp. 68-70,72.
- Fowler, Martin, *Analysis Patterns: Reusable Object Models*, Addison-Wesley, 1997.
- Papurt, David M., "The Object Model: Attribute and Association," *Report on Object Analysis & Design*, Volume 1, Number 4, November - December 1994, pp. 14-17.
- Rumbaugh, James, "A Search for Values: Attributes and Associations," *Journal of Object-Oriented Programming*, Volume 9, Number 3, June, 1996, pp. 6-8,49.
- Tanzer, Christian, "Remarks on Object-Oriented Modeling of Associations," *Journal of Object-Oriented Programming*, Volume 7, Number 9, February, 1995, pp. 43-46.

Name

Associations for Analysis

Context

You are creating an analysis model. You have already applied *Deferred Attributes*, or are applying it simultaneously to this pattern. As in *Deferred Attributes*, you are trying to retain the expressive power of links without diminishing that power through clutter.

Because you are doing analysis, you cannot be certain of the scope of your target applications. Even if you could be certain, you may think it unwise to limit your analysis to the target applications, for this lessens the possibilities of reuse. You may even be a member of one of several teams working in the same domain, teams that hope to share results and thereby reduce individual efforts.

Problem

Which eventual attributes are best modeled as associations during analysis?

Forces

- Analysis models meet different needs from design models, just as analysis meets different needs from design.

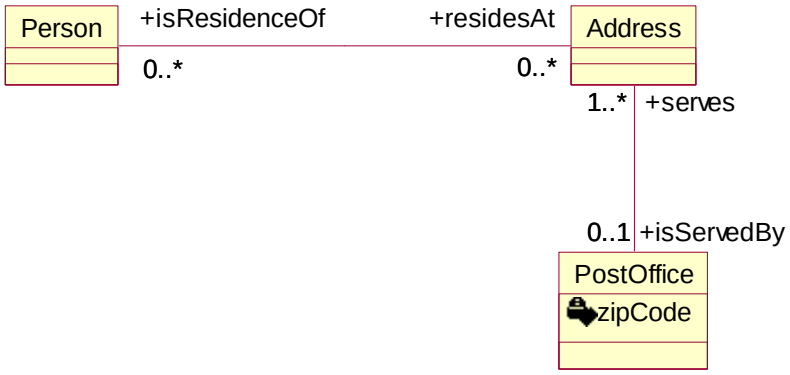
- A properly constructed analysis model contains elements conceptually important to its domain, even if these elements may be suspected of being unnecessary to an application under development. You suppress conceptually important but unnecessary elements during design, not analysis.
- According to Fowler 1997, "A number of object-oriented practitioners are uncomfortable with using associations in OO analysis. They see associations as violating the OO programming principle of encapsulation."
- A properly constructed analysis model represents its domain, not implementation of an application in that domain. You do not address programming principles in analysis.
- Navigability is generally unimportant in analysis. Some writers prefer associations in analysis models to be bidirectional (e.g. Fowler and Scott 1997: "each association has two **roles**"; see also Papurt 1994). Others prefer them to be non-directional (e.g. Henderson-Sellers 1998). UML equivalently represents both unstated and bidirectional navigability: "If navigability has not been decided, then it is bidirectional in the general case" [Rumbaugh et al. 1999].

Solution

1. If navigation from one element of a domain to another conceptually goes in both directions, model the relationship as an association. This will enable the analysis to be reused by applications to which either direction matters, even if one direction is not relevant to your application. For example, in a domain in which addresses may potentially be used for more than mailing, model an association between address and person during analysis, even if you will make address an attribute of person in design. In a demographics domain, model an association between ZIP code (i.e., post office) and address.
2. If "the reverse direction [of a candidate association] is determined to be unimportant *during analysis*" [Papurt 1994; his emphasis] – that is, if the navigation is conceptually unidirectional – normally model the element as an attribute. For example, in a domain in which addresses are used only to create mailing labels, post office should be modeled as a class with an association to address on only one diagram, following the solution in *Deferred Attributes* (item 4).
3. Do not let programming principles and implementation issues drive analysis. During analysis, strive to represent the domain.

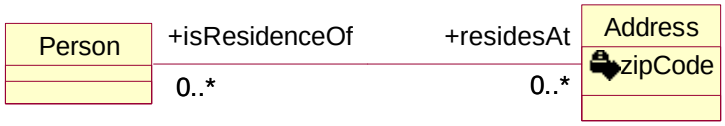
Examples

In certain demographic domains, a post office is more than a part of a mailing address. In some insurance applications, post office (or Zip code) is used to set rates. In some marketing applications, it is used to identify potential customers. In such applications, this would be an appropriate diagram:



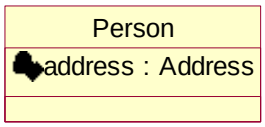
(created in UML with Rational Rose 98)

In other domains, post office is simply a part of an address. In some such domains, this would be an appropriate diagram:



(created in UML with Rational Rose 98)

Finally, there are domains in which addresses are used for nothing but mailing labels. In such domains, address would normally be diagrammed as an attribute of type Address:



(created in UML with Rational Rose 98i)

Resulting Context

This pattern helps create models that cannot normally be used for design without modification. For such modifications, see *Attributes for Design*.

References

- Fowler, Martin, *Analysis Patterns: Reusable Object Models*, Addison-Wesley, 1997.
- Fowler, Martin, with Kendall Scott, *UML Distilled: Applying the Standard Object Modeling Language*, Addison-Wesley, 1997.

- Henderson-Sellers, Brian, "Open Relationships - Associations, Mappings, Dependencies, and Uses," *Journal of Object-Oriented Programming*, Volume 10, Number 9, February 1998, pp. 49-57.
- Papurt, David M., "The Object Model: Attribute and Association," *Report on Object Analysis & Design*, Volume 1, Number 4, November - December 1994, pp. 14-17.
- Rumbaugh, James, Ivar Jacobson, and Grady Booch, *The Unified Modeling Language Reference Manual*, Addison-Wesley, 1999.

Name

Attributes for Design

Context

You are creating a design model. You have already applied *Deferred Attributes*, or are applying it simultaneously to this pattern. As in *Deferred Attributes*, you are trying to retain the expressive power of links without diminishing that power through clutter.

Because you are doing design, you should have received the products of analysis. If so, you plan to select those elements necessary to design the application under development, and add information required in moving towards implementation. Because analysis has a broader scope than design, you expect to winnow out elements or their aspects. In this process, you may leave some elements previously modeled as associations as such, while reducing others to attributes. You will undergo a related process even if your design commences without benefit of analysis. Instead of winnowing, you will be creating, but your decisions should look towards the same results.

Problem

Which eventual attributes are best modeled as associations during design?

Forces

- Design models meet different needs from analysis models, just as design meets different needs from analysis.
- A design model often eliminates elements that do not take part in an application under development, even if they are conceptually important to the domain.
- The concerns of object-oriented practitioners that associations violate the programming principle of encapsulation (Fowler 1997), which were dismissed in *Associations for Analysis*, are germane to design.
- Navigability is important in design. In design, “associations represent responsibilities” [Fowler and Scott 1997], and responsibilities are directed, either unidirectionally or bidirectionally.

Solution

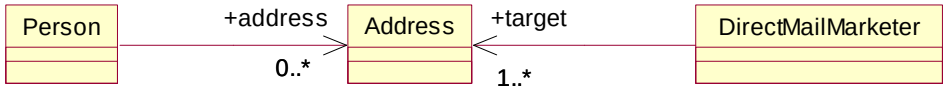
1. If navigation from one element of a domain to another conceptually goes in both directions (as shown in an analysis model), but one direction is irrelevant to the application being designed, normally model the relationship as a unidirectional association in design. (See Papurt 1994.) The association between a person and his or her car may be bidirectional or have unstated directionality in an analysis model, for example, but be navigable only from owner to car in a design model. The association would nonetheless remain an association.
2. If all that you retain of an association is a value, generally model it as an attribute in design. This applies both to associations that were unidirectional in analysis and to associations that were reduced to unidirectionality in design (for which see item 1 above). For example, if an application uses address only for mailing purposes, you should normally model it as an attribute, even if address was an associated class in analysis. Following the solution of *Deferred Attributes* (item 4), there would then be only one diagram on which address appears as a class.

Examples

In most domains, this is an appropriate design model for the relationship between person and car:

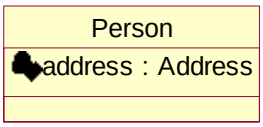
(created in UML with Rational Rose 98)

In a direct mail marketing domain, this design model appropriately represents the relationship between person and address:



(created in UML with Rational Rose 98)

In other domains, this would be appropriate for representing the relationship between person and addresss:



(created in UML with Rational Rose 98i)

References

- Fowler, Martin, *Analysis Patterns: Reusable Object Models*, Addison-Wesley, 1997.
- Fowler, Martin, with Kendall Scott, *UML Distilled: Applying the Standard Object Modeling Language*, Addison-Wesley, 1997.
- Papurt, David M., "The Object Model: Attribute and Association*," Report on Object Analysis & Design*, Volume 1, Number 4, November - December 1994, pp. 14-17.

Acknowledgements

The author thanks Kyle Brown, whose shepherding of this paper helped greatly in the uncovering of its contents.

Disclaimer

The views expressed in this paper are those of the author and do not necessarily reflect the position of the Federal Reserve Bank of New York or the Federal Reserve System.