Attributes and Associations in Object Modeling

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

Object-oriented software developers often create models, especially if they work cooperatively with users. Developers use models to show users that they understand their domain and needs, and to give programmers a basis for coding. Models that meet both these goals can be difficult to create.

Some of the trouble lies in determining whether to represent domain elements with attributes or associations. Superficially, these decisions may appear simple. 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). What could be easier?

When you apply this principle, though, you soon encounter tough cases. It is clear that a character is a value without identity. But this is less clear of 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 of 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 is in an insurance application.

Deciding between attributes and associations is a problem for modeling, not implementation. When you implement a model in an object-oriented language, each element of the model that you retain becomes part of a class specification. Code is not sand, and you cannot draw a line representing an association in it. This should create no misgivings: good code is accurate and efficient, but does not mirror a model.

In theory, you could build models without associations, just as you could build models without attributes. (For the first option, see Velho and Carapuça 1994; for the second, see Tanzer 1995, who rejects it.) You therefore cannot choose between an attribute and an association by appeal to accuracy, that is, by reference to the reality underlying the model. Instead, you choose one or the other for what it communicates to the users and to the programmers.

Determining what to model with an association and what with an attribute is thus an art, not a science. However, you can subject aspects of the determination to science, or at least to engineering. The following two patterns take the broad heuristics for identifying attributes and associations found in object modeling literature, and replace them with more mechanical decision procedures. Successful application of the patterns will reserve art for what is beyond science.

The first pattern, *Stable Representations*, applies uniformly to both analysis and design, and shows how to identify those elements that should always (with limited and specific exceptions) be modeled either with attributes or with associations. The second pattern, *Changeable Representations*, addresses forces that may yield different results for analysis and design, particularly regarding object navigation. You can apply the patterns sequentially, but need not. *Stable Representation* deals with easier cases, and you might want to apply it first for that reason.

These patterns deal with only one set of problems in object modeling. They do not, for example, show when to create association objects, which by definition "associat[e] two other objects" (Boyd 1998). These two patterns are therefore part of a larger and as yet unwritten pattern language.

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.

Acknowledgements

The author thanks Kyle Brown, whose shepherding of this paper helped greatly in the uncovering of its contents, and the participants in its workshop presentation at the PLoP 1999 conference. They were gracious, thoughtful, and thorough.

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.

Stable Representations

Context

You are creating a model. You know that any associations you draw will eventually be implemented with attributes (or not at all), but you also know that your model will lack 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.

In creating a model, you respond both to forces that remain the same whether you are engaged in analysis or design and to forces that may differ. This pattern resolves forces that remain the same. A second pattern, *Changeable Representations*, responds to forces that may differ for analysis and design, and deals with most of the issues surrounding object navigation.

Problem

Which domain elements are best modeled with attributes, and which with 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 a postal code in that domain other than what it is. An insurance domain, however, may associate postal code with risk level and use it for setting rates.
- Elements have different connections 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.

- Object models tend to ignore time. But objects hold some elements throughout their existence or nearly so, and hold other elements temporarily, or only at certain times. A person, for example, generally receives a name shortly after birth and has some name from that time on. On the other hand, a person may have a spouse, but generally not while a child, and perhaps not into old age. (See D'Souza 1994.)

- Different assumptions may be made about the visibility of elements when they are modeled with attributes or associations. The software tool Rational Rose, for example, treats attributes in models as private and associations in models as public by default. Unless a modeler changes element visibility, then, Rose will generate Java code implementing a model's attributes with private attributes and a model's associations with public attributes.

Solution

1. Use an attribute to model an element when

   - There is nothing to say about the element other than what it is, and nothing that you will ask it to do. An example is a character, such as the letter 'c'.
   - Everything that you will ask the element to do comes as part of any object-oriented language you might use. 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.

2. Use an attribute to model an element even though you define a class for it when

   - You treat the element as a fundamental type, as if it came with the language. An example is currency (Fowler 1997).
   - Only one class has access to the element. Most identifiers (e.g. name, social security number, and bank routing number) fall under this guideline.

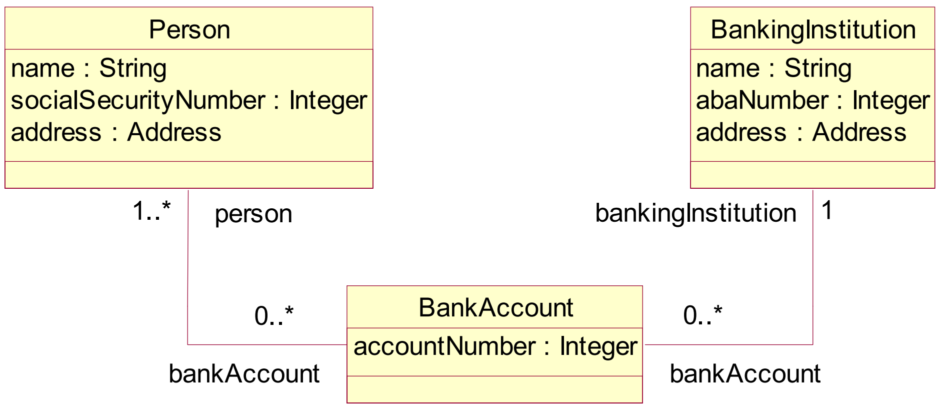
   Since you must code the fundamental types you create, you must model them on at least one diagram as classes with methods. Similarly, since elements with access through a single class often have methods defined for such operations as formatting and validation, you will normally model them also as classes on at least one diagram. (Bank routing numbers, for example, are validated by the computation of the last digit, which is a check digit.) Create one or more diagrams dedicated to your fundamental types, and one or more diagrams dedicated to elements with access through a single class. Associations may appear these diagrams (for example, between address and post office), but need not.

3. Use an association to model an element when

   - The element is not subject to the previous guidelines, and different classes in a model use it differently. An example is bank account, which person and banking institution use differently. Note the qualification "in a model," which recognizes that you 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, and therefore be subject to a previous guideline.
   - The element is part of a bi-directional relationship between two classes. (You need not model it as a bi-directional association, though: see *Changeable Representations*.) Do not model this element by specifying an attribute in each class of the type of the other class.
   - Objects do not hold the element throughout their lifetimes (or nearly so). 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 an element is 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. Address will appear as a class on another diagram, following solution #2. ABA (bank routing) number and the other integers may also be drawn as classes on another diagram if, for example, they include validation routines.



Resulting Context

While this pattern solves many of the issues in choosing between attributes and associations, it does not deal with situations where the different expressive goals of analysis and design models come into play. You should therefore apply *Changeable Representations* simultaneously to or after applying this pattern. Choices not covered by either of these patterns remain subject to art.

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.

Changeable Representations

Context

You are creating a model, either for analysis or design. You have already applied *Stable Representations*, or are applying it simultaneously to this pattern. In applying this pattern, you are resolving forces that differ for analysis and design. As in *Stable Representations*, you are trying to retain the expressive power of links without diminishing that power through clutter.

Problem

Which domain elements are best modeled with attributes, and which with associations, in analysis and in design?

Forces

- While it is inevitable that some programming considerations will effect a model (such as the decision to implement in an object-oriented language), such considerations should have as limited a place in analysis as is feasible.

- Analysis should (and often does) precede the delineation of target applications. Even when it does not, directing analysis to target applications lessens the possibility of reuse.

- An analysis model should represent its domain, and include elements conceptually important to that domain even if those elements are thought unnecessary to target applications.

- When design follows analysis, the designer adapts those products of analysis that are necessary to target applications. A designer normally eliminates elements that have no part in target applications.

- 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." Object-oriented practitioners feel this discomfort even more deeply, and more appropriately, in design.

- Associations should not be modeled in only one direction during analysis, where navigability is in general unimportant.[1] In some cases, modeling an association in only one direction during analysis would cause no harm, but it would also realize no gain, so there is no reason to risk it.

- Navigability matters to design. In design, "associations represent responsibilities" (Fowler and Scott 1997), and responsibilities are directed, either unidirectionally or bidirectionally.

Solution

During analysis,

1. 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 – model the element as an attribute. For example, in a domain in which geographical areas and demographics are of no consequence, model post office as an attribute of address (although it may appear as a class with an association to address on one diagram, following *Stable Representations* solution #2).

2. If navigation from one element of a domain to another conceptually goes in both directions, model the relationship as an association. Do this even if one direction is not relevant to target application(s), for other applications will then be able to reuse the analysis. 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 post office (identified in the United States by ZIP code) and address.

During design,

1. If navigation from one element of a domain to another conceptually goes in both directions (as found in analysis), but one direction is irrelevant to the application(s) being designed, model the relationship as a unidirectional association. (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

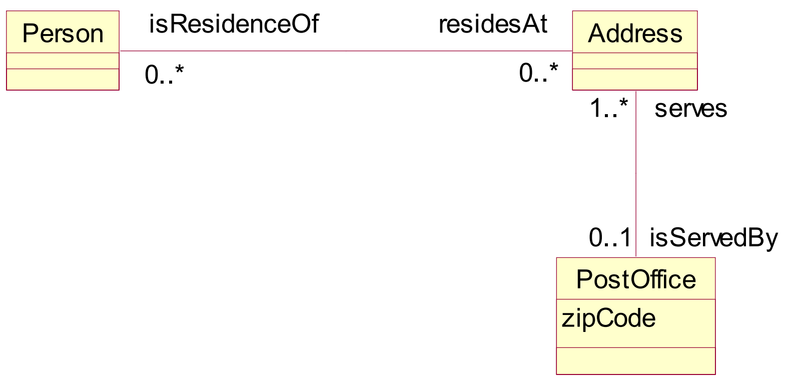
[1] Some writers prefer associations in analysis models to be bidirectional (e.g. Fowler and Scott 1997: "each association has two **roles**"; see also Papurt 1994), while others prefer them to be non-directional (e.g. Henderson-Sellers 1998). The distinction between bidirectional and non-directional associations has little consequence, however. UML even represents both unstated and bidirectional navigability equivalently: "If navigability has not been decided, then it is bidirectional in the general case" (Rumbaugh et al. 1999).

association modeled in analysis would nonetheless remain an association in design.

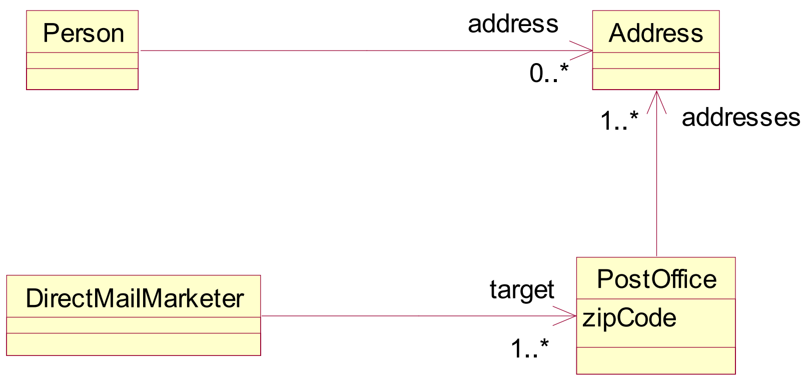
2. If, however, all that you retain of an element modeled as an association in analysis is a value, model it as an attribute. For example, if target application(s) use address only for mailing purposes, model address as an attribute, even if it was an associated class in analysis. (Following *Stable Relationships* solution #2, there will be one diagram on which address appears as a class, but there will be no associations to that class.)

Examples

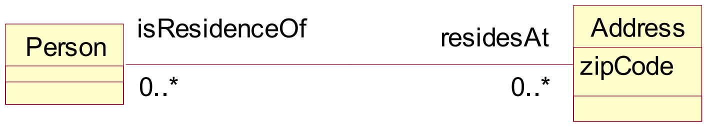
In certain demographic domains, a post office is more than a part of a mailing address. Some insurance applications, for example, use post office (ZIP code) to set rates, and some marketing applications use it to identify potential customers. In such applications, this would be an appropriate diagram in an analysis model:



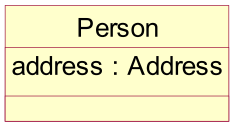
In a design model for a marketing application, this would be an appropriate diagram:



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



In other domains, addresses are used for nothing but mailing labels. In such domains, address would be appropriately diagrammed in both analysis and design models as an attribute of type Address (which would be specified on another diagram):



Resulting Context

When this pattern is applied in conjunction with *Stable Representations*, it solves those issues in choosing between attributes and associations that are subject to engineering. Other choices remain subject to art.

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.